

PROGRESS[®] ORBIX[®]

Orbix Programmer's Reference C++ Edition

Version 3.3, SP11 March 2012

Progress Orbix v3.3.11

© 2012 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Actional, Apama, Artix, Business Empowerment, Business Making Progress, Corticon, Corticon (and design), DataDirect (and design), DataDirect Connect, DataDirect Connect64, DataDirect Technologies, DataDirect XML Converters, DataDirect XQuery, DataXtend, Dynamic Routing Architecture, Empowerment Center, Fathom, Fuse Mediation Router, Fuse Message Broker, Fuse Services Framework, IONA, Making Software Work Together, Mindreef, ObjectStore, OpenEdge, Orbix, PeerDirect, Powered by Progress, PowerTier, Progress, Progress DataXtend, Progress Dynamics, Progress Business Empowerment, Progress Empowerment Center, Progress Empowerment Program, Progress OpenEdge, Progress Profiles, Progress Results, Progress Software Business Making Progress, Progress Software Developers Network, Progress Sonic, ProVision, PS Select, RulesCloud, RulesWorld, Savvion, SequeLink, Shadow, SOAPscope, SOAPStation, Sonic, Sonic ESB, SonicMQ, Sonic Orchestration Server, SpeedScript, Stylus Studio, Technical Empowerment, WebSpeed, Xcalia (and design), and Your Software, Our Technology-Experience the Connection are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. AccelEvent, Apama Dashboard Studio, Apama Event Manager, Apama Event Modeler, Apama Event Store, Apama Risk Firewall, AppsAlive, AppServer, ASPen, ASP-in-a-Box, BusinessEdge, Cache-Forward, CloudEdge, DataDirect Spy, DataDirect SupportLink, Fuse, FuseSource, Future Proof, GVAC, High Performance Integration, ObjectStore Inspector, ObjectStore Performance Expert, OpenAccess, Orbacus, Pantero, POSSE, ProDataSet, Progress Arcade, Progress CloudEdge, Progress Cloudware, Progress Control Tower, Progress ESP Event Manager, Progress ESP Event Modeler, Progress Event Engine, Progress RFID, Progress RPM, Progress Responsive Cloud, Progress Responsive Process Management, Progress Software, PSE Pro, SectorAlliance, SeeThinkAct, Shadow z/Services, Shadow z/Direct, Shadow z/Events, Shadow z/Presentation, Shadow Studio, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Sonic Business Integration Suite, Sonic Process Manager, Sonic Collaboration Server, Sonic Continuous Availability Architecture, Sonic Database Service, Sonic Workbench, Sonic XML Server, The Brains Behind BAM, WebClient, and Who Makes Progress are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

Third Party Acknowledgements: One or more products in the Progress Orbix v3.3.11 release includes third party components covered by licenses that require that the following documentation notices be provided:

Progress Orbix v3.3.11 incorporates OpenSSL/SSLey v0.9.8.i technology from OpenSSL.org. Such Technology is subject to the following terms and conditions: LICENSE ISSUES

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLey license apply to the toolkit. See below for the actual license texts. Actually both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL please contact openssl-core@openssl.org.

OpenSSL License

Copyright (c) 1998-2008 The OpenSSL Project. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:

"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)"

4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.
5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following acknowledgment:

"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).
This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Original SSLeay License

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved. This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscapes SSL. This library is free for commercial and non-commercial use as long as the following conditions are adhered to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

"This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)"
The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-).

4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement:

"This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Progress Orbix v3.3.11 incorporates mcpp v2.6.4 from SourceForge (<http://sourceforge.net/softwaremap/index.php>). Such technology is subject to the following terms and conditions: Copyright (c) 1998, 2002-2007 Kiyoshi Matsui kmatsui@t3.rim.or.jp All rights reserved. This software including the files in this directory is provided under the following license. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress Orbix v3.3.11 incorporates IDL Compiler Front End v1.0 from Sun Microsystems. Such technology is subject to the following terms and conditions: COPYRIGHT NOTICE on OMG IDL CFE: Copyright 1992 Sun Microsystems, Inc. Printed in the United States of America. All Rights Reserved. This product is protected by copyright and distributed under the following license restricting its use. The Interface Definition Language Compiler Front End (CFE) is made available for your use provided that you include this license and copyright notice on all media and documentation and the software program in which this product is incorporated in whole or part. You may copy and extend functionality (but may not remove functionality) of the Interface Definition Language CFE without charge, but you are not authorized to license or distribute it to anyone else except as part of a product or program developed by you or with the express written consent of Sun Microsystems, Inc. ("Sun"). The names of Sun Microsystems, Inc. and any of its subsidiaries or affiliates may not be used in advertising or publicity pertaining to distribution of Interface Definition Language CFE as permitted herein. This license is effective until terminated by Sun for failure to comply with this license. Upon termination, you shall destroy or return all code and documentation for the Interface Definition Language CFE. The Interface Definition Language CFE may not be exported outside the United States without first obtaining the appropriate government approvals. INTERFACE DEFINITION LANGUAGE CFE IS PROVIDED AS IS WITH NO WARRANTIES OF ANY KIND INCLUDING THE WARRANTIES OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR ARISING FROM A COURSE OF DEALING, USAGE OR TRADE PRACTICE. INTERFACE DEFINITION LANGUAGE CFE IS PROVIDED

WITH NO SUPPORT AND WITHOUT ANY OBLIGATION ON THE PART OF Sun OR ANY OF ITS SUBSIDIARIES OR AFFILIATES TO ASSIST IN ITS USE, CORRECTION, MODIFICATION OR ENHANCEMENT. SUN OR ANY OF ITS SUBSIDIARIES OR AFFILIATES SHALL HAVE NO LIABILITY WITH RESPECT TO THE INFRINGEMENT OF COPYRIGHTS, TRADE SECRETS OR ANY PATENTS BY INTERFACE DEFINITION LANGUAGE CFE OR ANY PART THEREOF. IN NO EVENT WILL SUN OR ANY OF ITS SUBSIDIARIES OR AFFILIATES BE LIABLE FOR ANY LOST REVENUE OR PROFITS OR OTHER SPECIAL, INDIRECT AND CONSEQUENTIAL DAMAGES, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19. Sun, Sun Microsystems and the Sun logo are trademarks or registered trademarks of Sun Microsystems, Inc. SunSoft, Inc. 2550 Garcia Avenue Mountain View, California 94043

Updated: 07-Mar-2012

Contents

Preface	25
Audience	25
Organization of this Guide	25
Document Conventions	26

Part I Orbix Class Reference

Memory Allocation	32
CORBA	35
CORBA::_OBJECT_TABLE_SIZE_DEFAULT	35
CORBA::arg()	35
CORBA::IT_chooseDefaultEnv()	36
CORBA::extract()	36
CORBA::insert()	37
CORBA::is_nil()	38
CORBA::ORB_init()	39
CORBA::release()	40
CORBA::string_alloc()	40
CORBA::string_dup()	41
CORBA::string_free()	41
CORBA::Any	43
CORBA::Any::Any()	46
CORBA::Any::Any()	46
CORBA::Any::Any()	46
CORBA::Any::~Any()	47
CORBA::Any::operator = ()	48
CORBA::Any::operator<< = ()	48
CORBA::Any::operator>> = ()	50
CORBA::Any::replace()	51
CORBA::Any::type()	52
CORBA::Any::value()	53

CORBA::AuthenticationFilter	55
CORBA::AuthenticationFilter::AuthenticationFilter()	55
CORBA::BOA	57
CORBA::BOA::activationMode	63
CORBA::BOA::anyClientsConnected()	63
CORBA::BOA::change_implementation()	63
CORBA::BOA::continueThreadDispatch()	64
CORBA::BOA::create()	64
CORBA::BOA::deactivate_impl()	65
CORBA::BOA::deactivate_obj()	66
CORBA::BOA::dispose()	66
CORBA::BOA::enableLoaders()	67
CORBA::BOA::filterBadConnectAttempts()	67
CORBA::BOA::filterBadConnectAttempts()	68
CORBA::BOA::get_id()	68
CORBA::BOA::getNoHangup()	69
CORBA::BOA::get_principal()	69
CORBA::BOA::getFileDescriptors()	71
CORBA::BOA::getFilter()	71
CORBA::BOA::impl_is_ready()	72
CORBA::BOA::isDeactivated()	74
CORBA::BOA::isEventPending()	75
CORBA::BOA::myActivationMode()	75
CORBA::BOA::myImplementationName()	75
CORBA::BOA::myImpRepPath()	76
CORBA::BOA::myIntRepPath()	76
CORBA::BOA::myMarkerName()	76
CORBA::BOA::myMarkerPattern()	77
CORBA::BOA::myMethodName()	77
CORBA::BOA::obj_is_ready()	77
CORBA::BOA::processEvents()	78
CORBA::BOA::processNextEvent()	80
CORBA::BOA::propagateTIEdelete()	80
CORBA::BOA::propagateTIEdelete()	81
CORBA::BOA::setImpl()	82
CORBA::BOA::setNoHangup()	83
CORBA::BOA::usingLoaders()	83
CORBA::Context	85
CORBA::Context::Context()	88

CORBA::Context::Context()	88
CORBA::Context::Context()	89
CORBA::Context::~Context()	89
CORBA::Context::_duplicate()	89
CORBA::Context::_nil()	90
CORBA::Context::context_name()	90
CORBA::Context::create_child()	90
CORBA::Context::delete_values()	91
CORBA::Context::get_count()	91
CORBA::Context::get_count_all()	91
CORBA::Context::get_values()	92
CORBA::Context::IT_create()	93
CORBA::Context::parent()	93
CORBA::Context::set_one_value()	94
CORBA::Context::set_values()	94
CORBA::ContextIterator	95
CORBA::ContextIterator::ContextIterator()	95
CORBA::ContextIterator::~ContextIterator()	95
CORBA::ContextIterator::operator()()	96
CORBA::DynamicImplementation	97
CORBA::DynamicImplementation::	
DynamicImplementation()	97
CORBA::DynamicImplementation::	
~DynamicImplementation()	98
CORBA::DynamicImplementation::invoke()	98
CORBA::Environment	99
CORBA::Environment::clear()	101
CORBA::Environment::_duplicate()	101
CORBA::Environment::Environment()	101
CORBA::Environment::Environment()	102
CORBA::Environment::Environment()	102
CORBA::Environment::Environment()	102
CORBA::Environment::~Environment()	103
CORBA::Environment::exception()	103
CORBA::Environment::exception()	104
CORBA::Environment::int()	104
CORBA::Environment::IT_create()	104
CORBA::Environment::_nil()	105

CORBA::Environment::operator=()	105
CORBA::Environment::operator=()	105
CORBA::Environment::operator=()	106
CORBA::Environment::request()	106
CORBA::Environment::request()	107
CORBA::Environment::timeout()	107
CORBA::Environment::timeout()	107
CORBA::Exception	109
CORBA::Exception::Exception()	109
CORBA::Exception::Exception()	109
CORBA::Exception::~~Exception()	110
CORBA::Exception::operator=()	110
CORBA::	
ExtraConfigFileCVHandler	111
CORBA::ExtraConfigFileCVHandler::	
ExtraConfigFileCVHandler()	112
CORBA::ExtraConfigFileCVHandler::	
~ExtraConfigFileCVHandler()	112
CORBA::ExtraRegistryCVHandler	113
CORBA::ExtraRegistryCVHandler::	
ExtraRegistryCVHandler()	114
CORBA::ExtraRegistryCVHandler::	
ExtraRegistryCVHandler()	114
CORBA::ExtraRegistryCVHandler::	
~ExtraRegistryCVHandler()	115
CORBA::ExtraRegistryCVHandler::GetRegKey()	115
CORBA::Filter	117
CORBA::Filter::Filter()	118
CORBA::Filter::~~Filter()	118
CORBA::Filter::inReplyFailure()	118
CORBA::Filter::inReplyPostMarshal()	119
CORBA::Filter::inReplyPreMarshal()	120
CORBA::Filter::inRequestPostMarshal()	120
CORBA::Filter::inRequestPreMarshal()	121
CORBA::Filter::outReplyFailure()	122
CORBA::Filter::outReplyPostMarshal()	123

CORBA::Filter::outReplyPreMarshal()	123
CORBA::Filter::outRequestPostMarshal()	124
CORBA::Filter::outRequestPreMarshal()	125
CORBA::Flags	127
CORBA::Flags::Flags()	127
CORBA::Flags::Flags()	128
CORBA::Flags::Flags()	128
CORBA::Flags::operator=()	128
CORBA::Flags::clrf()	128
CORBA::Flags::isNil()	129
CORBA::Flags::isSet()	129
CORBA::Flags::isSetAll()	129
CORBA::Flags::isSetAny()	129
CORBA::Flags::ULong()	130
CORBA::Flags::reset()	130
CORBA::Flags::setArgDef()	130
CORBA::Flags::setf()	130
CORBA::ImplementationDef	131
CORBA::ImplementationDef::_duplicate()	131
CORBA::ImplementationDef::_nil()	132
CORBA::ImplementationDef::IT_create()	132
CORBA::IT_IOCallback	133
CORBA::IT_IOCallback::AtOrbixFDLowLimit()	134
CORBA::IT_IOCallback::ForeignFDExcept()	135
CORBA::IT_IOCallback::ForeignFDRead()	135
CORBA::IT_IOCallback::ForeignFDWrite()	135
CORBA::IT_IOCallback::OrbixFDClose()	136
CORBA::IT_IOCallback::OrbixFDOpen()	136
CORBA::IT_IOCallback::ResumeListeningBelowFDHigh()	136
CORBA::IT_IOCallback::StopListeningAtFDHigh()	137
CORBA::IT_reqTransformer	139
CORBA::IT_reqTransformer::free_buf()	140
CORBA::IT_reqTransformer::transform()	141
CORBA::IT_reqTransformer::transform_error()	142
CORBA::IT_reqTransformer::setRemoteHost()	143

CORBA::LoaderClass	145
CORBA::LoaderClass::LoaderClass()	146
CORBA::LoaderClass::~~LoaderClass()	146
CORBA::LoaderClass::load()	147
CORBA::LoaderClass::record()	148
CORBA::LoaderClass::rename()	149
CORBA::LoaderClass::save()	149
CORBA::NamedValue	151
CORBA::NamedValue::NamedValue()	152
CORBA::NamedValue::NamedValue()	152
CORBA::NamedValue::~~NamedValue()	153
CORBA::NamedValue::IT_create()	153
CORBA::NamedValue::_duplicate()	154
CORBA::NamedValue::flags()	154
CORBA::NamedValue::name()	154
CORBA::NamedValue::_nil()	155
CORBA::NamedValue::operator=()	155
CORBA::NamedValue::value()	155
CORBA::NullLoaderClass	157
CORBA::NullLoaderClass::NullLoaderClass()	157
CORBA::NullLoaderClass::record()	158
CORBA::NVList	159
CORBA::NVList::NVList()	161
CORBA::NVList::NVList()	161
CORBA::NVList::NVList()	162
CORBA::NVList::~~NVList()	162
CORBA::NVList::operator=()	162
CORBA::NVList::_duplicate()	162
CORBA::NVList::_nil()	163
CORBA::NVList::add()	163
CORBA::NVList::add_item()	163
CORBA::NVList::add_value()	164
CORBA::NVList::add_item_consume()	164
CORBA::NVList::add_value_consume()	165
CORBA::NVList::count()	165
CORBA::NVList::IT_create()	166
CORBA::NVList::item()	166
CORBA::NVList::remove()	167

CORBA::NVLIterator	169
CORBA::NVLIterator::NVLIterator()	169
CORBA::NVLIterator::NVLIterator()	170
CORBA::NVLIterator::operator()()	170
CORBA::NVLIterator::setList()	170
CORBA::Object	171
CORBA::Object::Object()	176
CORBA::Object::Object()	176
CORBA::Object::Object()	176
CORBA::Object::Object()	176
CORBA::Object::~~Object()	177
CORBA::Object::_attachPost()	177
CORBA::Object::_attachPre()	178
CORBA::Object::_closeChannel()	179
CORBA::Object::_create_request()	179
CORBA::Object::_deref()	180
CORBA::Object::_duplicate()	181
CORBA::Object::_enableInternalLock()	182
CORBA::Object::_fd()	183
CORBA::Object::_get_implementation()	183
CORBA::Object::_get_interface()	184
CORBA::Object::_getPost()	184
CORBA::Object::_getPre()	184
CORBA::Object::_hash()	185
CORBA::Object::_hasValidOpenChannel()	185
CORBA::Object::_host()	186
CORBA::Object::_implementation()	186
CORBA::Object::_interfaceHost()	187
CORBA::Object::_interfaceImplementation()	187
CORBA::Object::_interfaceMarker()	187
CORBA::Object::_is_a()	187
CORBA::Object::_is_equivalent()	188
CORBA::Object::_isNull()	189
CORBA::Object::_isNullProxy()	189
CORBA::Object::_isRemote()	190
CORBA::Object::_loader()	190
CORBA::Object::_marker()	191
CORBA::Object::_marker()	191
CORBA::Object::_nil()	192
CORBA::Object::_non_existent()	192
CORBA::Object::_object_to_string()	192

CORBA::Object::_refCount()	193
CORBA::Object::_request()	193
CORBA::Object::_save()	194

CORBA::ORB **195**

CORBA::ORB::abortSlowConnects()	207
CORBA::ORB::abortSlowConnects()	207
CORBA::ORB::ActivateCVHandler()	208
CORBA::ORB::ActivateOutputHandler()	208
CORBA::ORB::addForeignFD()	208
CORBA::ORB::addForeignFDSet()	209
CORBA::ORB::baseInterfacesOf()	209
CORBA::ORB::bindUsingIOP()	210
CORBA::ORB::bindUsingIOP()	210
CORBA::ORB::BOA_init()	211
CORBA::ORB::closeChannel()	212
CORBA::ORB::collocated()	212
CORBA::ORB::collocated()	213
CORBA::ORB::connectionTimeout()	214
CORBA::ORB::connectionTimeout()	214
CORBA::ORB::create_environment()	215
CORBA::ORB::create_list()	215
CORBA::ORB::create_named_value()	216
CORBA::ORB::create_operation_list()	216
CORBA::ORB::DeactivateCVHandler()	217
CORBA::ORB::DeactivateOutputHandler()	217
CORBA::ORB::DEFAULT_TIMEOUT	218
CORBA::ORB::defaultRxTimeout()	218
CORBA::ORB::defaultRxTimeout()	219
CORBA::ORB::defaultTxTimeout()	219
CORBA::ORB::defaultTxTimeout()	220
CORBA::ORB::eagerListeners()	220
CORBA::ORB::eagerListeners()	221
CORBA::ORB::getAllOrbixFDs()	222
CORBA::ORB::getForeignFDSet()	222
CORBA::ORB::GetConfigValue()	222
CORBA::ORB::GetConfigValueBool()	223
CORBA::ORB::GetConfigValueLong()	223
CORBA::ORB::getDiagnostics()	224
CORBA::ORB::getMyReqTransformer()	224
CORBA::ORB::getSelectableFDSet()	224
CORBA::ORB::get_default_context()	225

CORBA::ORB::get_next_response()	225
CORBA::ORB::INFINITE_TIMEOUT	226
CORBA::ORB::isBaseInterfaceOf()	226
CORBA::ORB::isDefaultRxTimeoutSet()	227
CORBA::ORB::isForeignFD()	227
CORBA::ORB::isOrbixFD()	228
CORBA::ORB::isOrbixSelectableFD()	228
CORBA::ORB::list_initial_services()	229
CORBA::ORB::makeIOR()	229
CORBA::ORB::makeOrbixObjectKey()	230
CORBA::ORB::maxConnectRetries()	230
CORBA::ORB::maxConnectRetries()	231
CORBA::ORB::mustRedefineDeref()	231
CORBA::ORB::mustRedefineDeref()	232
CORBA::ORB::myHost()	232
CORBA::ORB::myServer()	232
CORBA::ORB::noReconnectOnFailure()	233
CORBA::ORB::noReconnectOnFailure()	234
CORBA::ORB::object_to_string()	234
CORBA::ORB::optimiseProtocolEncoding()	235
CORBA::ORB::optimiseProtocolEncoding()	236
CORBA::ORB::Output()	236
CORBA::ORB::pingDuringBind()	237
CORBA::ORB::pingDuringBind()	237
CORBA::ORB::PlaceCVHandlerAfter()	238
CORBA::ORB::PlaceCVHandlerBefore()	239
CORBA::ORB::poll_next_response()	239
CORBA::ORB::registerIOCallback()	240
CORBA::ORB::registerIOCallbackObject()	241
CORBA::ORB::registerPerObjectServiceContext Handler()	242
CORBA::ORB::registerPerRequestServiceContext Handler()	243
CORBA::ORB::ReinitialiseConfig()	243
CORBA::ORB::removeForeignFD()	244
CORBA::ORB::removeForeignFDSet()	244
CORBA::ORB::reSizeObjectTable()	245
CORBA::ORB::resolve_initial_references()	245
CORBA::ORB::resortToStatic()	246
CORBA::ORB::resortToStatic()	247
CORBA::ORB::send_multiple_requests_deferred()	247
CORBA::ORB::send_multiple_requests_oneway()	248
CORBA::ORB::SetConfigValue()	249

CORBA::ORB::SetConfigValueBool()	250
CORBA::ORB::SetConfigValueLong()	250
CORBA::ORB::setDiagnostics()	251
CORBA::ORB::setMyReqTransformer()	251
CORBA::ORB::setReqTransformer()	252
CORBA::ORB::setServerName()	252
CORBA::ORB::set_unsafeDelete()	253
CORBA::ORB::set_unsafeFDClose()	253
CORBA::ORB::string_to_object()	254
CORBA::ORB::string_to_object()	255
CORBA::ORB::supportBidirectionalIIOP()	256
CORBA::ORB::supportBidirectionalIIOP()	257
CORBA::ORB::unregisterIOCallbackObject()	257
CORBA::ORB::unregisterPerObjectServiceContext Handler()	258
CORBA::ORB::unregisterPerRequestServiceContext Handler()	258
CORBA::ORB::useHostNameInIOR()	259
CORBA::ORB::useRemoteIsACalls()	260
CORBA::ORB::useReverseLookup	260
CORBA::ORB::useServiceContexts()	261
CORBA::ORB::useTransientPort()	261
CORBA::ORB::useTransientPort()	262
CORBA::ORB::usingRemoteIsACalls()	262
CORBA::ORB::usingReverseLookup	263
CORBA::ORB::usingServiceContexts()	263
CORBA::Principal	265
CORBA::Principal::Principal()	265
CORBA::Principal::_duplicate()	266
CORBA::Principal::_nil()	266
CORBA::Principal::IT_create()	266
CORBA::Request	267
CORBA::Request::Request()	272
CORBA::Request::Request()	273
CORBA::Request::~~Request()	273
CORBA::Request::addToContextList()	273
CORBA::Request::getContextList()	274
CORBA::Request::operator>>()	274
CORBA::Request::operator<<()	275

CORBA::Request::_duplicate()	277
CORBA::Request::_nil()	277
CORBA::Request::arguments()	277
CORBA::Request::assumeArgsOwnership()	278
CORBA::Request::assumeResultOwnership()	278
CORBA::Request::ctx()	279
CORBA::Request::ctx()	279
CORBA::Request::decodeArray()	279
CORBA::Request::decodeStringOp()	280
CORBA::Request::decodeBndStrOp()	281
CORBA::Request::decodeInOutStrOp()	281
CORBA::Request::descriptor()	282
CORBA::Request::encodeArray()	282
CORBA::Request::encodeStringOp()	283
CORBA::Request::env()	283
CORBA::Request::extractOctet()	283
CORBA::Request::get_response()	284
CORBA::Request::insertOctet()	284
CORBA::Request::invoke()	285
CORBA::Request::IT_create()	285
CORBA::Request::operation()	286
CORBA::Request::poll_response()	286
CORBA::Request::reset()	287
CORBA::Request::result()	287
CORBA::Request::send_deferred()	287
CORBA::Request::send_oneway()	288
CORBA::Request::setOperation()	289
CORBA::Request::set_return_type()	290
CORBA::Request::setTarget()	290
CORBA::Request::target()	290
CORBA::ServerRequest	291
CORBA::ServerRequest::ServerRequest()	293
CORBA::ServerRequest::~~ServerRequest()	293
CORBA::ServerRequest::arguments()	294
CORBA::ServerRequest::ctx()	294
CORBA::ServerRequest::exception()	294
CORBA::ServerRequest::env()	295
CORBA::ServerRequest::env()	295
CORBA::ServerRequest::op_def()	295
CORBA::ServerRequest::op_name()	296
CORBA::ServerRequest::operation()	296

CORBA::ServerRequest::params()	297
CORBA::ServerRequest::result()	297
CORBA::ServerRequest::target()	298
CORBA::ServiceContextHandler	299
CORBA::ServiceContextHandler::ServiceContext Handler()	299
CORBA::ServiceContextHandler::~~ServiceContext Handler()	300
CORBA::ServiceContextHandler::incomingRequest()	300
CORBA::ServiceContextHandler::outboundRequest()	301
CORBA::ServiceContextHandler::incomingReply()	301
CORBA::ServiceContextHandler::outboundReply()	302
CORBA::ServiceContextHandler::context_id()	303
CORBA::String_var	305
CORBA::String_var::String_var()	306
CORBA::String_var::String_var()	306
CORBA::String_var::String_var()	306
CORBA::String_var::~~String_var()	306
CORBA::String_var::operator=()	307
CORBA::String_var::operator[]()	307
CORBA::String_var::char*()	307
CORBA::SystemException	309
CORBA::SystemException::SystemException()	310
CORBA::SystemException::SystemException()	310
CORBA::SystemException::SystemException()	311
CORBA::SystemException::~~SystemException()	311
CORBA::SystemException::operator=()	311
CORBA::SystemException::operator<<()	311
CORBA::SystemException::_narrow()	312
CORBA::SystemException::completed()	312
CORBA::SystemException::completed()	312
CORBA::CompletionStatus	313
CORBA::SystemException::minor()	313
CORBA::SystemException::minor()	313
CORBA::ThreadFilter	315
CORBA::ThreadFilter::ThreadFilter()	316

CORBA::TypeCode	317
CORBA::TypeCode::TypeCode()	321
CORBA::TypeCode::TypeCode()	322
CORBA::TypeCode::~~TypeCode()	322
CORBA::TypeCode::operator=()	322
CORBA::TypeCode::operator==(())	322
CORBA::TypeCode::operator!=(())	323
CORBA::TypeCode::_duplicate()	323
CORBA::TypeCode::_nil()	323
CORBA::TypeCode::equal()	324
CORBA::TypeCode::IT_create()	324
CORBA::TypeCode::kind()	324
CORBA::TypeCode::param_count()	325
CORBA::TypeCode::parameter()	325
CORBA::UserCVHandler	327
CORBA::UserCVHandler::UserCVHandler()	328
CORBA::UserCVHandler::~~UserCVHandler()	328
CORBA::UserCVHandler::GetValue()	328
CORBA::UserException	331
CORBA::UserException::UserException()	331
CORBA::UserException::UserException()	332
CORBA::UserException::operator=()	332
CORBA::UserException::_narrow()	332
CORBA::UserOutput	333
CORBA::UserOutput::UserOutput()	334
CORBA::UserOutput::~~UserOutput()	334
CORBA::UserOutput::Output()	334

Part II

IDL Interface to the Interface Repository

Common CORBA Data Types	337
CORBA::DefinitionKind	337
CORBA::Identifier	337
CORBA::RepositoryId	338
CORBA::ScopedName	338
CORBA::AliasDef	339
AliasDef::describe()	339
AliasDef::original_type_def	340
CORBA::ArrayDef	341
ArrayDef::element_type	341
ArrayDef::element_type_def	342
ArrayDef::length	342
CORBA::AttributeDef	343
AttributeDef::describe()	343
AttributeDef::mode	344
AttributeDef::type	344
AttributeDef::type_def	345
CORBA::ConstantDef	347
ConstantDef::describe()	347
ConstantDef::type	348
ConstantDef::type_def	348
ConstantDef::value	349
CORBA::Contained	351
Contained::absolute_name()	352
Contained::containing_repository()	352
Contained::defined_in	352
Contained::describe()	353

Contained:: id	354
Contained:: move ()	354
Contained:: name	355
Contained:: version	355
CORBA::Container	357
Container:: contents()	359
Container:: create_alias()	360
Container:: create_constant()	360
Container:: create_enum()	361
Container:: create_exception()	362
Container:: create_interface()	363
Container:: create_module()	363
Container:: create_struct()	364
Container:: create_union()	365
Container:: describe_contents()	366
Container:: lookup()	366
Container:: lookup_name()	367
CORBA::EnumDef	369
EnumDef:: describe()	369
EnumDef:: members	370
CORBA::ExceptionDef	371
ExceptionDef:: describe()	371
ExceptionDef:: members	372
ExceptionDef:: type	373
CORBA::IDLType	375
IDLType:: type	375
CORBA::IObject	377
IObject:: def_kind	377
IObject:: destroy()	377
CORBA::IT_Repository	379
IT_Repository:: start()	379
IT_Repository:: commit()	380
IT_Repository:: rollBack()	380

IT_Repository::active_transactions()	380
CORBA::ModuleDef	381
ModuleDef::describe()	381
CORBA::OperationDef	383
OperationDef::contexts	384
OperationDef::exceptions	384
OperationDef::describe()	384
OperationDef::mode	385
OperationDef::params	385
OperationDef::result	386
OperationDef::result_def	386
CORBA::PrimitiveDef	387
PrimitiveDef::kind	387
CORBA::Repository	389
Repository::create_array()	390
Repository::create_sequence()	390
Repository::create_string()	391
Repository::get_primitive()	391
Repository::describe_contents()	391
Repository::lookup_id()	392
CORBA::SequenceDef	393
SequenceDef::bound	393
SequenceDef::element_type	394
SequenceDef::element_type_def	394
SequenceDef::type	394
CORBA::StringDef	395
StringDef::bound	395
CORBA::StructDef	397
StructDef::describe()	397
StructDef::members	398

CORBA::TypedefDef	399
TypedefDef::describe()	399
CORBA::UnionDef	401
UnionDef::describe()	401
UnionDef::discriminator_type	402
UnionDef::discriminator_type_def	402
UnionDef::members	403

Part III

IDL Interface to the Orbix Daemon

IDL Interface to the Orbix Daemon	407
IT_daemon::addDirRights()	410
IT_daemon::addInvokeRights()	410
IT_daemon::addInvokeRightsDir()	411
IT_daemon::addLaunchRights()	411
IT_daemon::addLaunchRightsDir()	411
IT_daemon::addMethod()	412
IT_daemon::addSharedMarker()	412
IT_daemon::addUnsharedMarker()	412
IT_daemon::changeOwnerServer()	413
IT_daemon::deleteDirectory()	413
IT_daemon::deleteServer()	413
IT_daemon::getServer()	414
IT_daemon::killServer()	414
IT_daemon::LaunchStatus	415
IT_daemon::listActiveServers()	415
IT_daemon::listServers()	415
IT_daemon::newDirectory()	416
IT_daemon::newPerMethodServer()	416
IT_daemon::newSharedServer()	417
IT_daemon::newUnSharedServer()	418
IT_daemon::removeDirRights()	418

<code>IT_daemon::removeInvokeRights()</code>	419
<code>IT_daemon::removeInvokeRightsDir()</code>	419
<code>IT_daemon::removeLaunchRights()</code>	419
<code>IT_daemon::removeLaunchRightsDir()</code>	420
<code>IT_daemon::removeMethod()</code>	420
<code>IT_daemon::removeSharedMarker()</code>	420
<code>IT_daemon::removeUnsharedMarker()</code>	421
<code>IT_daemon::serverDetails</code>	421
<code>IT_daemon::serverExists()</code>	422

Appendices

Appendix A	
IDL Reference	425
Appendix B	
System Exceptions	433

Preface

The *Orbix Programmer's Reference C++ Edition* provides a complete reference for the application programming interface (API) to Orbix.

Audience

The *Orbix Programmer's Reference C++ Edition* is designed as a reference for Orbix programmers. Before using this guide, read the *Orbix C++ Programmer's Guide* to learn about writing distributed applications using Orbix.

Orbix documentation is periodically updated. New versions between releases are available at this site:

<http://communities.progress.com/pcom/docs/DOC-105220>

If you need assistance with Orbix or any other PROGRESS products, contact PROGRESS at <http://www.progress.com/orbix/orbix-support.html>. Comments on PROGRESS documentation can be sent to

<http://www.progress.com/en/about/contact.html>.

Organization of this Guide

This guide is divided into three parts as follows:

Part I, Orbix Class Reference

This part provides a full reference listing for each of the Orbix C++ classes. These classes are defined in the Orbix include file `CORBA.h` and provide the main application programming interface to Orbix.

Part II, IDL Interface to the Interface Repository

The Interface Repository is the component of Orbix that provides runtime access to IDL definitions. The application programming interface to this component is defined in IDL. Part II provides an exhaustive reference for the IDL interface to the Interface Repository.

Part III, IDL Interface to the Orbix Daemon

The Orbix daemon process, `orbixd`, manages several components of Orbix, including the Orbix Implementation Repository. This part provides a complete reference for the IDL interface to the Orbix daemon, which allows you to access the daemon functionality in your Orbix applications. The Orbix daemon acts as an Orbix server with server name `IT_daemon`.

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 `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.

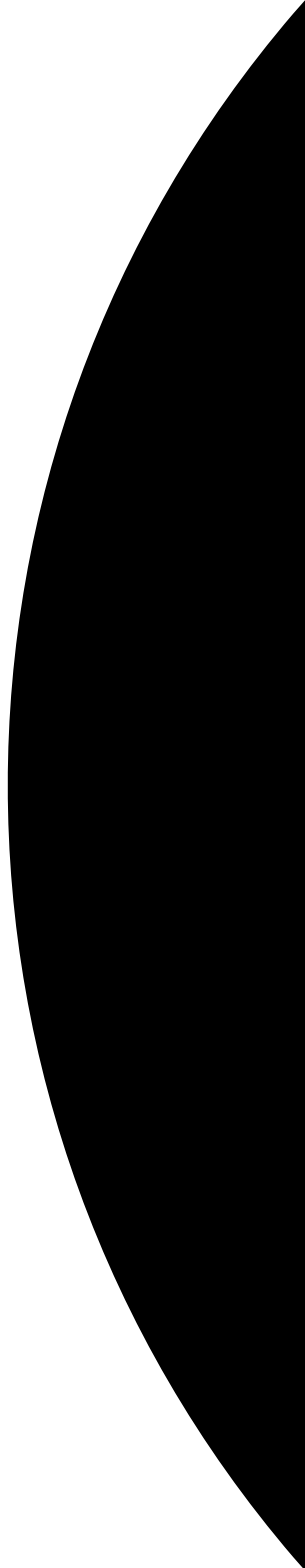
This guide may use the following keying conventions:

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 number sign represents the UNIX command shell prompt for a command that requires root privileges.
>	The notation > represents the DOS, Windows NT, or Windows 95 command prompt.
...	A horizontal or vertical ellipsis 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.

Part I

Orbix Class Reference



The CORBA.h Classes

The Orbix include file `CORBA.h` implements the IDL CORBA module defined by the Object Management Group (OMG). This module contains a number of IDL interfaces and pseudo interfaces that are mapped to C++ classes as described in the *Orbix C++ Programmer's Guide*. Orbix adds member functions to these classes and also provides additional classes to implement Orbix features such as *loaders* and *filters*. To assist programmers, each member function in these classes is labelled “CORBA compliant” or “Orbix specific” as appropriate.

`CORBA.h` also contains member functions that are needed only by generated code, by older versions of Orbix, or internally by Orbix. Since Orbix programmers should not need to use such functions—and indeed are recommended not to use them since they may not be supported in future releases—these are not documented in this guide.

Memory Allocation

This section highlights the general rules for memory management that are followed in the `CORBA.h` classes. Unless stated otherwise, you can assume the following:

Copy constructor	<p>Example:</p> <pre> // C++ class T { T(const T& t); }; </pre> <p>Usage:</p> <pre> // C++ T t; T new_t = t; </pre> <p>Initializes a new class object from an existing one. For a class <code>T</code>, the copy constructor creates a new <code>T</code> and deep copies its argument <code>t</code>.</p>
Assignment operator	<p>Example:</p> <pre> // C++ class T { operator=(const T& t); }; </pre> <p>Usage:</p> <pre> // C++ T t1, t2; t1 = t2; </pre> <p>The assignment operator frees memory associated with the target (<code>t1</code>) object and deep copies its argument (<code>t2</code>) to the invoked object.</p>
const argument	<p>Behaves as an “in” parameter. The caller is responsible for freeing any dynamically allocated memory.</p>

non-const argument	Behaves as an “out” or “inout” parameter. Assume the function modifies the object or pointer.
const return value	Orbix is responsible for freeing any dynamically allocated memory. The returned value should be copied if the caller wishes to retain it.
non-const return value	The caller is responsible for freeing the object.
<u>ptr</u> return value	The caller is responsible for freeing the object reference or assigning it to a <u>var</u> variable for automatic management.

CORBA

Synopsis The CORBA namespace implements the IDL CORBA module and includes a number of classes and other definitions specific to Orbix.

This chapter describes the functions and some useful definitions described directly in the CORBA namespace. Classes defined in the CORBA module are described in their individual chapters.

CORBA::_OBJECT_TABLE_SIZE_DEFAULT

Synopsis

```
static const CORBA::ULong _OBJECT_TABLE_SIZE_DEFAULT;
```

Description

The default size of the object table. All Orbix objects (including proxies) in an address space are registered in its object table (OT), a hash table that maps from object identifiers to the location of objects in virtual memory. If the table contains many objects, overflow chains are automatically added by Orbix.

You can change the default size (which is of the order of 1000) using `CORBA::ORB::reSizeObjectTable()`.

Notes

Orbix specific.

See Also

```
CORBA::ORB::reSizeObjectTable()
```

CORBA::arg()

Synopsis

```
static IT_Request_LS arg(const char* name);
```

Description

A manipulator function to assist in inserting arguments into a `CORBA::Request`, by naming the argument explicitly. For example:

```
// C++
// Insert parameter "height".
// Here, r is a CORBA::Request.
r << CORBA::arg("height") << 65;
```

Explicit naming of parameters does not remove the requirement that parameters must be inserted in the proper order. However, if the same name is used again, its previous value is replaced with a new value.

Notes Orbix specific.

See Also `CORBA::Request::operator<<()`

CORBA::IT_chooseDefaultEnv()

Synopsis `static CORBA::Environment, CORBA::IT_chooseDefaultEnv();`

Description The default environment. Each function of an IDL C++ class has a default parameter whose value is set to `CORBA::IT_chooseDefaultEnv()`. Refer to class `CORBA::Environment` for details.

The `CORBA::Environment` object returned by `CORBA::IT_chooseDefault_Env()` used in the Orbix API and in IDL C++ classes is subject to change. However, any change does not affect application programmers who should continue to use `CORBA::IT_chooseDefaultEnv()` as the default value for the `CORBA::Environment` parameter of implementation class operations.

Notes Orbix specific.

See Also `CORBA::Environment`

CORBA::extract()

Synopsis `static IT_Request_RS extract(
 const char* tcode, void* type);`

Description A manipulator function to extract a user-defined IDL type from a `CORBA::Request` object.

An example of its use for structs is:

```
// IDL
struct Example {
    long l;
    char c;
};
```

```
// C++
CORBA::Request r;
Example e;
r >> CORBA::extract(_tc_Example, &e);
```

`CORBA::extract()` uses the `TypeCode` generated by the IDL compiler for the type. In this case, `_tc_Example` is the `TypeCode` for the IDL struct `Example`.

This manipulator also works for primitive types and for arrays.

Parameters

- `tcode` The `TypeCode` object reference for the type of the second parameter. The type of this parameter in the `extract()` manipulator is `char*`. (For historical reasons the underlying implementation is in terms of `char*`.) An appropriate conversion takes place if you pass a `TypeCode` object reference.
- `type` A pointer to the user-defined type.

Notes

Orbis specific. The CORBA compliant function is `CORBA::Request::result()`.

See Also

`CORBA::insert()`
`CORBA::Request::result()`
`CORBA::Request::operator>>()`
`CORBA::Request()`
`CORBA::TypeCode`

CORBA::insert()

Synopsis

```
static IT_Request_LS CORBA::insert(
    const char* _tc_string, void* type);
static IT_Request_LS CORBA::insert(
    const char* tcode, void* type, Flags flags);
```

Description

A manipulator function to insert a user-defined IDL type into a `CORBA::Request` object.

An example of its use for structs is:

```
// IDL
struct Example {
    long l;
    char c;
```

```
};  
  
// C++  
CORBA::Request r;  
Example e;  
e.l = 27; e.c = 'f';  
r << CORBA::insert(_tc_Example,  
                   &e, CORBA::inMode);
```

`CORBA::insert()` uses the `TypeCode` generated by the IDL compiler for the type. In this case, `_tc_Example` is the `TypeCode` for the IDL struct `Example`.

This manipulator also works for primitive types and for arrays.

Parameters

- `tcode` The `TypeCode` object reference for the type of the second parameter. The type of this parameter in the `extract()` manipulator is `char*`. (For historical reasons the underlying implementation is in terms of `char*`.) An appropriate conversion takes place if you pass a `TypeCode` object reference.
- `type` A pointer to the user-defined type.
- `flags` The parameter passing mode: `CORBA::inMode`, `CORBA::outMode` or `CORBA::inoutMode`.

Notes

Orbix specific.

See Also

`CORBA::extract()`
`CORBA::TypeCode`
`CORBA::Request::operator<<()`
`CORBA::Request::insertOctet()`
`CORBA::Request::encodeArray()`

CORBA::is_nil()

Synopsis

```
static CORBA::Boolean CORBA::is_nil(IDL_Interface_ptr obj) const;
```

Description

A version of this function is generated for each IDL interface, `IDL_Interface`, and for each pseudo object type.

The function tests if `obj` is a nil reference.

Return Value	Returns 1 (TRUE) if <code>obj</code> is a nil object reference, returns 0 (FALSE) otherwise.
Notes	CORBA compliant.
See Also	<code>CORBA::Object::_isNullProxy()</code> <code>CORBA::Object::_isNull()</code>

CORBA::ORB_init()

Synopsis

```
CORBA::ORB_ptr CORBA::ORB_init(int& argc,
    char** argv,
    ORBid orb_identifier,
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description Initializes a client or server's connection to Orbix. In Orbix, the object reference returned by `ORB_init()` is identical to that in `CORBA::Orbix`.

Code using the `ORB_init()` function must be linked with the Orbix library.

On UNIX platforms, this library is named `liborbix` (`liborbixmt` for multi-threaded Orbix). On OpenVMS platforms, this library is named `liborbix.olb` (`liborbixmt.olb` for multi-threaded Orbix). On OS/390 this library is named `ORBIXhlq.MT.LIB.ORBIX`. On Windows NT, the library is named `ITCi.lib` (`ITMi.lib` for multi-threaded Orbix).

Parameters

<code>argc</code>	The number of arguments in <code>argv</code> .
<code>argv</code>	A sequence of option or configuration strings used if <code>orb_identifier</code> is a null string. Each string is of the form: <code>-ORB<suffix> <value></code> where <code><suffix></code> is the name of the option being set, and <code><value></code> is the value to which the option is set. Any string that is not in this format is ignored. An example parameter to identify the Orbix ORB is: <code>-ORBid Orbix</code>

`orb_identifier` A string identifying the ORB. The string “Orbix” identifies the Orbix ORB from PROGRESS Software. (Names of ORBs are locally administered by ORB vendors rather than by the OMG.)

If this parameter is null, the content of `argv` is checked.

Notes CORBA compliant. In Orbix, it is not necessary to call this function before using the ORB since Orbix automatically initializes a client or server's connection, making access to the ORB available through the `CORBA::Orbix` object.

See Also `CORBA::ORB::BOA_init()`

CORBA::release()

Synopsis

```
static void CORBA::release(IDL_Interface_ptr obj);
```

Description A version of this function is generated for each IDL interface type, `IDL_Interface`, and for each pseudo object type.

The function decrements the reference count of `obj`. The object is freed by Orbix if the reference count is then zero.

Calling `release()` on a nil object reference has no effect.

Notes CORBA compliant.

See Also `CORBA::A::_duplicate()`
`CORBA::Object::_refCount()`

CORBA::string_alloc()

Synopsis

```
static char* CORBA::string_alloc(CORBA::ULong len);
```

Description Dynamically allocates a string of length `len+1`. A conforming program should use this function to dynamically allocate a string that is passed between a client and a server.

Return Value Returns a pointer to the start of the character array; returns a zero pointer if it cannot perform the allocation.

Notes CORBA compliant.

See Also `CORBA::string_free()`
`CORBA::string_dup()`

CORBA::string_dup()

Synopsis `static char* CORBA::string_dup(const char* s);`

Description Duplicates the string `s`.

Return Value Returns a duplicate of the string `s`; returns a zero pointer if it is unable to perform the duplication. `CORBA::string_alloc()` may be used to allocate space for the string.

Notes CORBA compliant.

See Also `CORBA::string_alloc()`
`CORBA::string_free()`

CORBA::string_free()

Synopsis `static void CORBA::string_free(char* str);`

Description Deallocates the string `str`. The string `str` must have been allocated using `CORBA::string_alloc()`.

Notes CORBA compliant.

See Also `CORBA::string_alloc()`
`CORBA::string_dup()`

CORBA::Any

Synopsis

The C++ class `CORBA::Any` implements the IDL basic type `any`, which allows the specification of values that can express an arbitrary IDL type. This allows a program to handle values whose types are not known at compile time. The IDL type `any` is most often used in code that uses the Interface Repository or the Dynamic Invocation Interface (DII).

Consider the following interface:

```
// IDL
interface Example {
    void op(in any value);
};
```

A client can construct an `any` to contain an arbitrary type of value and then pass this in a call to operation `op()`. A process receiving an `any` must determine what type of value it stores and then extract the value.

Orbix

Type `any` is mapped to a C++ class that conceptually contains a `TypeCode` and a value:

```
// C++
// In namespace CORBA.
class Any {
public:
    Any();
    Any(const Any&);
    Any(TypeCode_ptr type, void* val,
        Boolean release = 0,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
    Any& operator = (const Any& a);
    ~Any();

    void operator<< = (CORBA::Short);
    void operator<< = (CORBA::Long);
    void operator<< = (CORBA::LongLong);
    void operator<< = (CORBA::UShort);
    void operator<< = (CORBA::ULong);
    void operator<< = (CORBA::ULongLong);
    void operator<< = (Float);
```

```
void operator<< = (Double);
void operator<< = (const char*);
void operator<< = (const Any&);
void operator<< = (const TypeCode_ptr&);
void operator<< = (Object_ptr);

Boolean operator>> = (CORBA::Short&) const;
Boolean operator>> = (CORBA::Long&) const;
Boolean operator>> = (CORBA::LongLong) const;
Boolean operator>> = (CORBA::UShort&) const;
Boolean operator>> = (CORBA::ULong&) const;
Boolean operator>> = (CORBA::ULongLong) const;
Boolean operator>> = (Float&) const;
Boolean operator>> = (Double&) const;
Boolean operator>> = (char*&) const;
Boolean operator>> = (Any&) const;
Boolean operator>> = (TypeCode_ptr&) const;
Boolean operator >> = (Object_ptr&) const;

void operator<< = (from_boolean b);
void operator<< = (from_octet o);
void operator<< = (from_char c);

Boolean operator>> = (to_boolean b) const;
Boolean operator>> = (to_octet o) const;
Boolean operator>> = (to_char c) const;

TypeCode_ptr type () const;
void* value () const;
void replace(TypeCode_ptr type, void* val,
             Boolean release = 0,
             CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

// Helper types needed for insertion of
// boolean, octet, and char:1
struct from_boolean {
    from_boolean(CORBA::Boolean b) : val(b) {};
    CORBA::Boolean val;
};
```

1. See “CORBA::Any::operator<< = ()” on page 48.

```
struct from_octet {
    from_octet(CORBA::Octet o) : val(o) {};
    CORBA::Octet val;
};

struct from_char {
    from_char(CORBA::Char c) : val(c) {};
    CORBA::Char val;
};

// Helper types needed to extract boolean,
// octet, and char:2
struct to_boolean {
    to_boolean(CORBA::Boolean& b) : ref(b) {};
    CORBA::Boolean& ref;
};

struct to_octet {
    to_octet(CORBA::Octet& o) : ref(o) {};
    CORBA::Octet& ref;
};

struct to_char {
    to_char(CORBA::Char& c) : ref(c) {};
    CORBA::Char& ref;
};
};
```

Notes CORBA compliant.

See Also CORBA::TypeCode

2. See “CORBA::Any::operator>>= ()” on page 50.

CORBA::Any::Any()

- Synopsis** `CORBA::Any::Any();`
- Description** The default constructor creates an `Any` with a `TypeCode` of type `tk_null` and with a zero value. The easiest and the type-safe way to construct an `Any` is to use the default constructor and then use `operator<< = ()` to insert a value into the `Any`. For example,
- ```
// C++
CORBA::Short s = 10;
CORBA::Any a;
a << = s;
```
- Notes** CORBA compliant.
- See Also** `CORBA::Any::operator<< = ()`  
Other `Any` constructors.

### CORBA::Any::Any()

- Synopsis** `CORBA::Any::Any(const CORBA::Any& a);`
- Description** Copy constructor. The constructor duplicates the `TypeCode_ptr` of `a` and copies the value.
- Notes** CORBA compliant.
- See Also** Other `Any` constructors.

### CORBA::Any::Any()

- Synopsis** `CORBA::Any::Any(CORBA::TypeCode_ptr type, void* val, CORBA::Boolean release = 0);`
- Description** Constructs an `Any` with a specific `TypeCode` and value. This constructor is needed for cases where it is not possible to use the default constructor and `operator<< = ()`. For example, since all strings are mapped to `char*`, it is not possible to create an `Any` with a specific `TypeCode` for a bounded string.

---

**Note:** This constructor is not type-safe; you must ensure consistency between the `TypeCode` and the actual type of the argument `val`.

---

### Parameters

- `type` A reference to a `CORBA::TypeCode`. The constructor duplicates this object reference.
- `value` The value pointer. A conforming program should make no assumptions about the lifetime of the value passed in this parameter once it has been passed to this constructor with `release = 1`.
- `release` A boolean variable to decide ownership of the storage pointed to by `val`. If set to 1, the `Any` object assumes ownership of the storage. If the `release` parameter is set to 0 (the default), the calling program is responsible for managing the memory pointed to by `val`.

**Notes** CORBA compliant.

**See Also** `CORBA::Any::operator<< = ()`  
`CORBA::Any::replace()`  
Other `Any` constructors.

### CORBA::Any::~~Any()

**Synopsis** `CORBA::Any::~~Any();`

**Description** Destructor for an `Any`. Depending on the value of the Boolean `release` parameter to the complex constructor, it frees the value contained in the `Any` based on the `TypeCode` of the `Any`. It then frees the `TypeCode`.

**Notes** CORBA compliant.

**See Also** `CORBA::Any::Any(CORBA::TypeCode_ptr, void*, Boolean)`

### **CORBA::Any::operator = ()**

|                    |                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <code>CORBA::Any::Any&amp; operator = (const CORBA::Any::Any&amp; a);</code>                                                                                                                  |
| <b>Description</b> | The assignment operator releases its <code>TypeCode</code> and frees the value if necessary; it duplicates the <code>TypeCode</code> of <code>a</code> and deep copies the parameter's value. |
| <b>Notes</b>       | CORBA compliant.                                                                                                                                                                              |

### **CORBA::Any::operator<< = ()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <pre>void operator&lt;&lt; = (CORBA::Short); void operator&lt;&lt; = (CORBA::Long); void operator&lt;&lt; = (CORBA::LongLong); void operator&lt;&lt; = (CORBA::UShort); void operator&lt;&lt; = (CORBA::ULong); void operator&lt;&lt; = (CORBA::ULongLong); void operator&lt;&lt; = (Float); void operator&lt;&lt; = (Double); void operator&lt;&lt; = (const char*); // Unbounded string. void operator&lt;&lt; = (const Any&amp;); void operator&lt;&lt; = (const TypeCode_ptr&amp;); void operator&lt;&lt; = (Object_ptr); void operator&lt;&lt; = (from_boolean); void operator&lt;&lt; = (from_octet); void operator&lt;&lt; = (from_char);</pre>                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Description</b> | <p>Inserts a value of the indicated type into an <code>Any</code>.</p> <p>Any previous value held by the <code>Any</code> will be properly deallocated. Each <code>operator&lt;&lt; = ()</code> takes a copy of the value being inserted (in the case of an object reference, <code>_duplicate()</code> is used). The <code>Any</code> is then responsible for memory management of the copy.</p> <p>The insertion function <code>operator&lt;&lt; = (const char* s)</code> assumes that the value passed in <code>s</code> is an unbounded string. A bounded string must be inserted into an existing <code>Any</code> using the function <code>CORBA::Any::replace()</code>.</p> <p>The C++ mapping for IDL types <code>boolean</code>, <code>octet</code> and <code>char</code> cannot be distinguished for the purpose of function overloading. Therefore, the 'helper' types <code>CORBA::Any::from_boolean</code>, <code>CORBA::Any::from_octet</code>, and <code>CORBA::Any::from_char</code> serve to distinguish these types.</p> |



You can use this as follows:

```
// C++
CORBA::Octet o = 030;
CORBA::Any a;

// To insert an octet into an Any:
a << = CORBA::Any::from_octet(o);

// An octet cannot be inserted as follows:
a << = o; // This will not compile.
```

An attempt to insert an unsigned char value into an Any results in a compile-time error.

To insert a user-defined type into an Any, the IDL source file must be compiled with the `-A` switch. An appropriate `operator<< = ()` is then generated from the IDL definition. For example, for the definition

```
// IDL
struct AStruct {
 string str;
 float number;
};
```

the following operator is generated:

```
// C++
void operator<< = (CORBA::Any& a, const AStruct& t);
```

This can be used as follows:

```
// C++
CORBA::Any a;
AStruct s;
// Initialise s.
a << = s;
```

**Parameters** `operator<< = (const char* s)` copies its parameter, `s`; `operator<< = (CORBA::Object_ptr t)` duplicates its object reference, `t`.

**Notes** CORBA compliant.

**See Also** `CORBA::Any::replace()`

### **CORBA::Any::operator>> = ()**

#### **Synopsis**

```
CORBA::Boolean operator>> = (CORBA::Short&) const;
CORBA::Boolean operator>> = (CORBA::Long&) const;
CORBA::Boolean operator>> = (CORBA::LongLong);
CORBA::Boolean operator>> = (CORBA::UShort&) const;
CORBA::Boolean operator>> = (CORBA::ULong&) const;
CORBA::Boolean operator>> = (CORBA::ULongLong);
CORBA::Boolean operator>> = (CORBA::Float&) const;
CORBA::Boolean operator>> = (CORBA::Double&) const;
CORBA::Boolean operator>> = (char*&) const;
CORBA::Boolean operator>> = (Any&) const;
CORBA::Boolean operator>> = (TypeCode_ptr&) const;
CORBA::Boolean operator>> = (Object_ptr&) const;
CORBA::Boolean operator>> = (to_boolean) const;
CORBA::Boolean operator>> = (to_octet) const;
CORBA::Boolean operator>> = (to_char) const;
```

#### **Description**

Extracts a value of the indicated type from an `Any`. You can determine the type of an `Any` by calling the member function `CORBA::Any::type()`, and you can extract the value using `operator>> = ()`.

You cannot distinguish the C++ mapping for IDL types `boolean`, `octet` and `char` for the purpose of function overloading. Therefore, the ‘helper’ types `CORBA::Any::to_boolean`, `CORBA::Any::to_octet`, and `CORBA::Any::to_char` serve to distinguish these types. You can use this as follows:

```
// C++
CORBA::Octet o;
CORBA::Any a = ...;

// How to extract an octet from an Any.
if (a >> = CORBA::Any::to_octet(o)) ...

// An octet cannot be extracted as follows:
if (a >> = o) ... // This will not compile.
```

An attempt to extract an unsigned `char` value from an `Any` results in a compile-time error.

To extract a user-defined type from an `Any`, you must compile the IDL source file with the `-A` switch. An appropriate `operator>> = ()` is then generated from the IDL definition.

For example, the definition:

```
// IDL
struct Details {
 string name;
};
```

results allows struct `Details` to be extracted as follows:

```
// C++
CORBA::Any a;
Details* d;
if(a >> = d) {
 ...
}
```

If the extraction is successful, the caller's pointer, `d`, points to a copy of the value inserted into the `Any`, and `operator>> = ()` returns 1. Note that:

- ◆ The `Any` is responsible for the memory management of the value. The caller must not try to delete or otherwise release this storage.
- ◆ The caller should not use the storage after the `Any` has been deallocated.
- ◆ You should avoid using `_var` types with the extraction operators, because they try to assume ownership of the storage owned by the `Any`.

**Return Value** These operators return 1 (`TRUE`) if the `Any` contains a value of the appropriate type; otherwise they return 0 (`FALSE`) (and set their parameter to an appropriate zero value). The value 0 is also returned if the `Any` does not contain a value.

**Notes** CORBA compliant.

## CORBA::Any::replace()

**Synopsis** `void replace(CORBA::TypeCode_ptr type, void* val,  
CORBA::Boolean release = 0);`

**Description** This member function is needed for cases where it is not possible to use `operator<< = ()` to insert into an existing `Any`. For example, because all strings are mapped to `char*`, it is not possible to create an `Any` with a specific `TypeCode` for a bounded string.

---

**Note:** This function is not type-safe; you must ensure consistency between the `TypeCode` and the actual type of the argument `val`.

---

### Parameters

- `type` A reference to a `CORBA::TypeCode`. The function duplicates this object reference.
- `value` The value pointer. A conforming program should make no assumptions about the lifetime of the value passed in this parameter if it has been passed to `Any::replace()` with `release = 1`.
- `release` A boolean variable to decide ownership of the storage pointed to by `val`. If set to 1, the `Any` object assumes ownership of the storage. If the `release` parameter is set to 0 (the default), the calling program is responsible for managing the memory pointed to by `val`.

**Notes** CORBA compliant.

**See Also** `CORBA::Any::operator<< = ()`

### **CORBA::Any::type()**

**Synopsis** `CORBA::TypeCode_ptr CORBA::Any::type() const;`

**Description** Returns a reference to the `TypeCode` associated with the `Any`.

**Return Value** The caller must release the reference when it is no longer needed, or assign it to a `TypeCode_var` variable for automatic management.

**Notes** CORBA compliant.

**See Also** `CORBA::Any::operator<< = ()`  
`CORBA::Any::operator>> = ()`

**CORBA::Any::value()**

**Synopsis**      `void* value() const;`

**Description**      Returns a pointer to the actual value stored in the `Any`. The exact nature of the returned value depends on the type of the value as shown below:

| IDL Type                         | value()                          |
|----------------------------------|----------------------------------|
| void                             | 0 (zero)                         |
| boolean                          | CORBA::Boolean*                  |
| char                             | CORBA::Char*                     |
| octet                            | CORBA::Octet*                    |
| short                            | CORBA::Short*                    |
| unsigned short                   | CORBA::UShort*                   |
| long                             | CORBA::Long*                     |
| unsigned long                    | CORBA::ULong*                    |
| long long                        | CORBA::LongLong*                 |
| unsigned long long               | CORBA::ULongLong*                |
| float                            | CORBA::Float*                    |
| double                           | CORBA::Double*                   |
| any                              | CORBA::Any*                      |
| Object                           | CORBA::Object_ptr*               |
| TypeCode                         | CORBA::TypeCode_ptr*             |
| NamedValue                       | CORBA::NamedValue_ptr*           |
| Object reference of interface I. | I_ptr*                           |
| InterfaceDescription             | CORBA::InterfaceDescription_ptr* |
| OperationDescription             | CORBA::OperationDescription_ptr* |

## Orbix Programmer's Reference C++ Edition

---

| IDL Type                             | value()                              |
|--------------------------------------|--------------------------------------|
| AttributeDescription                 | CORBA::AttributeDescription_ptr*     |
| ParameterDescription                 | CORBA::ParameterDescription_ptr*     |
| RepositoryDescription                | CORBA::RepositoryDescription_ptr*    |
| ModuleDescription                    | CORBA::ModuleDescription_ptr*        |
| ConstDescription                     | CORBA::ConstDescription_ptr*         |
| ExceptionDescription                 | CORBA::ExceptionDescription_ptr*     |
| TypeDescription                      | CORBA::TypeDescription_ptr*          |
| FullInterfaceDescription             | CORBA::FullInterfaceDescription_ptr* |
| Sequences of any of the above types. | Pointer to sequence.                 |
| struct                               | Pointer to struct.                   |
| string                               | char**                               |
| fixed                                | Pointer to fixed.                    |
| array                                | Pointer to array slice.              |

**Notes** CORBA compliant.

**See Also** CORBA::Any::type()

# CORBA::AuthenticationFilter

**Synopsis** CORBA::AuthenticationFilter is a derived class of class CORBA::Filter; it is used to pass authentication information between processes. The default implementation transmits the name of the principal (user name) to the server when the channel between the client and the server is first established and adds it to all requests at the server side.

You can override the default authentication filter by declaring a derived class of CORBA::AuthenticationFilter and then creating an instance of this class.

The authentication filter is always the first filter in a filter chain.

**Orbix**

```
// C++
class AuthenticationFilter : public CORBA::Filter {
 AuthenticationFilter();
};
```

**Notes** Orbix specific.

**See Also** CORBA::Filter  
CORBA::ThreadFilter

## CORBA::AuthenticationFilter::AuthenticationFilter()

**Synopsis** CORBA::AuthenticationFilter();

**Description** You cannot create direct instances of AuthenticationFilter: the constructor is protected to enforce this.

**Notes** Orbix specific.





# CORBA::BOA

## Synopsis

Class `CORBA::BOA` is a derived class of `CORBA::ORB` that implements the OMG CORBA BOA pseudo interface, and adds a number of functions specific to Orbix. BOA stands for “Basic Object Adapter”.

`CORBA::BOA` provides functions that control Orbix from the server. These include functions to:

- ◆ Activate and deactivate servers.
- ◆ Activate and deactivate objects.
- ◆ Create and interpret object references.

The functions on this class are invoked through the `CORBA::Orbix` object on the server; this is a static object of class `CORBA::BOA`.

## CORBA

```
// Pseudo IDL
pseudo interface BOA (
 Object create(in ReferenceData id,
 in InterfaceDef intf,
 in ImplementationDef impl);
 void dispose(in Object obj);
 ReferenceData get_id(in Object obj);
 void change_implementation(in Object obj,
 in ImplementationDef impl);
 Principal get_principal(in Object obj,
 in Environment env);
 void impl_is_ready(in Object obj,
 in ImplementationDef impl);
 void deactivate_impl(
 in ImplementationDef impl);
 void obj_is_ready(in ImplementationDef impl);
 void deactivate_obj(in Object obj);
};
```

```
Orbix // C++
 typedef _IDL_SEQUENCE_octet ReferenceData;

 class BOA : public ORB {

 public:

 BOA();

 virtual ~BOA();

 Boolean propagateTIEdelete() const;

 Boolean filterBadConnectAttempts() const;

 Boolean usingLoaders() const;

 Boolean isDeactivated() const;

 Object_ptr create(
 const ReferenceData& id,
 InterfaceDef_ptr intf,
 ImplementationDef_ptr impl,
 Environment& IT_env =
 IT_chooseDefaultEnv());

 ReferenceData* get_id(
 Object_ptr obj,
 Environment& IT_env =
 IT_chooseDefaultEnv());

 void dispose(
 Object_ptr obj,
 Environment& IT_env =
 IT_chooseDefaultEnv());

 void change_implementation(
 Object_ptr obj,
 ImplementationDef_ptr impl,
 Environment& IT_env =
 IT_chooseDefaultEnv());
```

---

```
Principal_ptr get_principal(
 Object_ptr obj,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Principal_ptr get_principal(
 Environment& IT_env =
 IT_chooseDefaultEnv());

ORBStatus processNextEvent(
 ULong timeOut = DEFAULT_TIMEOUT,
 Environment& IT_env =
 IT_chooseDefaultEnv());

ORBStatus processEvents(
 ULong timeOut = DEFAULT_TIMEOUT,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean isEventPending(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

Boolean anyClientsConnected() const;

Boolean setNoHangup(Boolean newMode);

Boolean getNoHangup() const;

fd_set getFileDescriptors() const;

void impl_is_ready(
 ImplementationDef_ptr serverName = "",
 ULong timeOut = DEFAULT_TIMEOUT,
 Environment& IT_env =
 IT_chooseDefaultEnv());

void impl_is_ready(
 ImplementationDef_ptr serverName,
 Environment& IT_env =
 IT_chooseDefaultEnv());

void impl_is_ready(
```

```
Environment& IT_env =
 IT_chooseDefaultEnv());

void deactivate_impl(
 ImplementationDef_ptr serverName,
 Environment& IT_env =
 IT_chooseDefaultEnv());

void obj_is_ready(
 Object_ptr obj,
 ImplementationDef_ptr impl,
 Environment& IT_env =
 IT_chooseDefaultEnv());

void obj_is_ready(
 Object_ptr obj,
 ImplementationDef_ptr impl,
 ULong timeOut = DEFAULT_TIMEOUT,
 Environment& IT_env =
 IT_chooseDefaultEnv());

void deactivate_obj(
 Object_ptr obj,
 Environment& IT_env =
 IT_chooseDefaultEnv());

void continueThreadDispatch(Request& req);

const char* myImpRepPath(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

const char* myIntRepPath(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

const char* myImplementationName(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

const char* myMarkerName(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;
```

```
const char* myMarkerPattern(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

const char* myMethodName(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

enum activationMode
{
 perMethodActivationMode,
 unsharedActivationMode,
 persistentActivationMode,
 sharedActivationMode,
 unknownActivationMode
};

activationMode myActivationMode(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

Boolean propagateTIEdelete(
 Boolean flag,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean filterBadConnectAttempts(
 Boolean flag,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean enableLoaders(
 Boolean flag,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Filter* getFilter();

static BOA_ptr _duplicate(
 BOA_ptr obj,
 Environment& IT_env =
 IT_chooseDefaultEnv());
```

```
static BOA_ptr _nil(
 Environment& IT_env =
 IT_chooseDefaultEnv());

static void setImpl(
 const char* interfaceName,
 DynamicImplementation& rDSISkeleton,
 Environment& IT_env =
 IT_chooseDefaultEnv());

static Object_ptr setImpl(
 const char* interfaceName,
 DynamicImplementation& rDSISkeleton,
 const char* pMarker,
 LoaderClass* pLoader = 0,
 Environment& IT_env =
 IT_chooseDefaultEnv());

};
```

**Notes** CORBA compliant.

**See Also** CORBA::ORB

---

## CORBA::BOA::activationMode

**Synopsis**

```
enum CORBA::BOA::activationMode {
 perMethodActivationMode,
 unsharedActivationMode,
 persistentActivationMode,
 sharedActivationMode,
 unknownActivationMode
};
```

**Description** Enumerates the activation modes for launching servers.

**Notes** Orbix specific.

**See Also** CORBA::BOA::myActivationMode()

## CORBA::BOA::anyClientsConnected()

**Synopsis** CORBA::Boolean CORBA::BOA::anyClientsConnected() const;

**Description** Determines if there are any connections from clients to the server.

**Return Value** Returns 1 (TRUE) if any clients are connected; returns 0 (FALSE) otherwise.

**Notes** Orbix specific.

## CORBA::BOA::change\_implementation()

**Synopsis**

```
void CORBA::BOA::change_implementation(CORBA::Object_ptr obj,
 ImplementationDef_ptr impl,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Changes the implementation (server name) associated with the object `obj`. You can use this function to overcome the problem of exporting an object reference from a persistent server before `impl_is_ready()` is called.

You can use the function `CORBA::ORB::setServerName()` to change the implementation for all objects created by a server.

---

**Note:** If a server creates an object and clients then invoke on this object, subsequent invocations on the object may fail following a call to `CORBA::BOA::change_implementation()` on that object.

---

### Parameters

`obj`     The object reference for which the implementation is to be changed.  
`impl`    The name of the new implementation (server).

**Notes**       CORBA compliant. This function is included for compliance with the CORBA specification. You are unlikely to need to use it.

**See Also**    `CORBA::BOA::impl_is_ready()`  
              `CORBA::ORB::setServerName()`

## **CORBA::BOA::continueThreadDispatch()**

### Synopsis

```
void CORBA::BOA::continueThreadDispatch(Request& req);
```

### Description

A per-process filter can create a thread to handle an incoming request. The function `continueThreadDispatch()` requests Orbix to continue processing request `req` in the context of a newly-created thread.

**Notes**       Orbix specific. This function requires Multi-Threaded Orbix (Orbix-MT).

**See Also**    `CORBA::Filter`  
              `CORBA::ThreadFilter`

## **CORBA::BOA::create()**

### Synopsis

```
CORBA::Object_ptr CORBA::BOA::create(
 const ReferenceData& id,
 InterfaceDef_ptr intf,
 CORBA::ImplementationDef_ptr impl,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

### Description

Creates a new object reference. Note that this function does not create an implementation object. As such, it makes little sense to use it unless an implementation object exists in the server or an appropriate loader is installed.



**Parameters**

|                   |                                                                                                                                                                        |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>id</code>   | Opaque identification information, supplied by the caller, and stored in an object. This is an IDL sequence of octets which, in Orbix, is mapped to the object marker. |
| <code>intf</code> | The Interface Repository object that specifies the set of interfaces implemented by the object.                                                                        |
| <code>impl</code> | The Implementation Repository entry (server name) that specifies the implementation to be used for the object.                                                         |

**Exceptions** If the `id` does not match the marker of an object currently resident (or loaded into) the server's address space, `CORBA::BOA::create()` raises a `CORBA::INV_OBJREF` exception.

**Notes** CORBA compliant. This function is included for compliance with the CORBA specification. You are unlikely to need to use it.

**See Also** `CORBA::Object::Object()`  
`CORBA::ORB::string_to_object()`  
`CORBA::LoaderClass`

**CORBA::BOA::deactivate\_impl()****Synopsis**

```
void CORBA::BOA::deactivate_impl(CORBA::ImplementationDef_ptr
impl, CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description**

A server that has called `impl_is_ready()` to indicate that it has completed initialization and is ready to receive requests, may subsequently indicate to Orbix that it wishes to discontinue receiving requests. It does so by calling `deactivate_impl()`, and passing the server name in the parameter `impl`. Calling `deactivate_impl()` causes `impl_is_ready()` to return.

---

**Note:** The `deactivate_impl()` function does not end the `impl_is_ready()` loop. It only ends the event loop on either an incoming or outgoing request. `deactivate_impl()` sets a flag that is checked once per event dispatch.

---

**Notes** CORBA compliant.

**See Also** `CORBA::BOA::impl_is_ready()`  
`CORBA::ImplementationDef`

### **CORBA::BOA::deactivate\_obj()**

**Synopsis**

```
void CORBA::BOA::deactivate_obj(CORBA::Object_ptr obj,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** A server (running in unshared activation mode) that has called `obj_is_ready()` to indicate that it has completed initialization and is ready to receive requests, may subsequently indicate to Orbix that it wishes to discontinue receiving requests for this object. It does so by calling `deactivate_obj()`, and passing the object whose marker caused the server process to be launched, in the parameter `obj`.

**Notes** CORBA compliant.

**See Also** `CORBA::BOA::obj_is_ready()`

### **CORBA::BOA::dispose()**

**Synopsis**

```
void CORBA::BOA::dispose(CORBA::Object_ptr obj,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;
```

**Description** Invalidates the object reference `obj`.

**Notes** CORBA compliant. This function is included for compliance with the CORBA specification. You are unlikely to need to use it.

**See Also** `CORBA::release()`

## CORBA::BOA::enableLoaders()

|                     |                                                                                                                                                                                                                                                                                                 |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>     | <pre>CORBA::Boolean CORBA::BOA::enableLoaders(CORBA::Boolean value, CORBA::Environment&amp; IT_env = CORBA::IT_chooseDefaultEnv());</pre>                                                                                                                                                       |
| <b>Description</b>  | It is occasionally useful to disable the loaders for a period. If, when binding to an object, the caller knows that the object is already loaded <i>if</i> it exists, it may be advisable to avoid involving the loaders if the object cannot be found.<br><br>By default, loaders are enabled. |
| <b>Parameters</b>   | <pre>value    1 (TRUE) enables loaders; 0 (FALSE) disables loaders.</pre>                                                                                                                                                                                                                       |
| <b>Return Value</b> | Returns the previous setting.                                                                                                                                                                                                                                                                   |
| <b>Notes</b>        | Orbix specific.                                                                                                                                                                                                                                                                                 |
| <b>See Also</b>     | <code>CORBA::LoaderClass</code>                                                                                                                                                                                                                                                                 |

## CORBA::BOA::filterBadConnectAttempts()

|                    |                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <pre>CORBA::Boolean CORBA::BOA::filterBadConnectAttempts( CORBA::Boolean value, CORBA::Environment&amp; IT_env = CORBA::IT_chooseDefaultEnv());</pre>                                                                                                                                                                                                                          |
| <b>Description</b> | By default, an exception is raised if a bad connection is made to a server waiting on the <code>CORBA::BOA::impl_is_ready()</code> , <code>CORBA::BOA::obj_is_ready()</code> , and <code>CORBA::BOA::processEvents()</code> functions. Such bad connection attempts can be caused, for example, by a connection attempt from an old version of Orbix (version 1.2 or earlier). |
| <b>Parameters</b>  | <pre>value    1 (TRUE) raises a CORBA::COMM_FAILURE exception if a bad connection is made to a server waiting on the CORBA::BOA::impl_is_ready(), CORBA::BOA::obj_is_ready(), and CORBA::BOA::processEvents() functions. This is the default. 0 (FALSE) Orbix silently handles such attempts without raising an exception.</pre>                                               |

**Return Value** Returns the previous setting.

**Notes** Orbix specific.

**See Also** `CORBA::BOA::filterBadConnectAttempts()`

### **CORBA::BOA::filterBadConnectAttempts()**

**Synopsis** `CORBA::Boolean CORBA::BOA::filterBadConnectAttempts() const;`

**Description** Returns the current setting of the `filterBadConnectAttempts` flag.

**Return Value**

- 1 (TRUE) Raises a `CORBA::COMM_FAILURE` exception if a bad connection is made to a server waiting on the `CORBA::BOA::impl_is_ready()`, `CORBA::BOA::obj_is_ready()`, and `CORBA::BOA::processEvents()` functions. This is the default.
- 0 (FALSE) Orbix silently handles such attempts without raising an exception.

**Notes** Orbix specific.

**See Also** `CORBA::BOA::filterBadConnectAttempts(CORBA::Boolean)`

### **CORBA::BOA::get\_id()**

**Synopsis** `ReferenceData* CORBA::BOA::get_id(CORBA::Object_ptr obj,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description** Returns the identification information of the object `obj` as set in `CORBA::BOA::create()`.

**Notes** CORBA compliant. This function is included for compliance with the CORBA specification. You are unlikely to need to use it.

**See Also** `CORBA::BOA::create()`  
`CORBA::Object::_marker()`

---

## CORBA::BOA::getNoHangup()

**Synopsis** `CORBA::Boolean CORBA::BOA::getNoHangup() const;`

**Description** Returns the current setting of the controlling noHangup flag.

**Return Value**

- 0 (FALSE) The functions `CORBA::BOA::impl_is_ready()`, `CORBA::BOA::obj_is_ready()`, and `CORBA::BOA::processEvents()` return if there are no events (operation calls, connections, disconnections) received by the server within the time-out period. This is true even if clients are currently connected to it. This is the default behaviour.
- 1 (TRUE) The functions `CORBA::BOA::impl_is_ready()`, `CORBA::BOA::obj_is_ready()`, and `CORBA::BOA::processEvents()` do not return while clients are connected to it. This is irrespective of any time-out period specified.

**Notes** Orbix specific.

**See Also** `CORBA::BOA::setNoHangup()`

## CORBA::BOA::get\_principal()

**Synopsis** `CORBA::Principal_ptr CORBA::BOA::get_principal(  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());  
CORBA::Principal_ptr get_principal(CORBA::Object_ptr ignored,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description** A server application can call either of these member functions when processing an operation call from a client. These member functions return the user name of the client process that made the current operation call.

**Parameters**

- ignored This parameter is ignored. It is supported for compatibility with CORBA.

`env` This must be the `Environment` passed to a server member function by Orbix. For example:

```
// C++
Account_ptr Bank_i::newAccount(const
char* name,
CORBA::Environment& pe) {
 ...
 char* principal =
CORBA::Orbix.get_principal(pe);
 ...
}
```

The following incorrect code returns a null pointer for the principal because the `env` parameter passed to `get_principal()` is not the `Environment` passed by the client:

```
// C++
Account_ptr Bank_i::newAccount(const
char* name,
CORBA::Environment& pe) {
 CORBA::Environment_ptr env;
 CORBA::create_environment(env);
 ...
 try { // Incorrect use of
 // get_principal().
 char* principal =
CORBA::Orbix.get_principal(env);
 }
 ...
}
```

**Notes** The function `get_principal(CORBA::Object_ptr ignored)` is CORBA compliant. The function `get_principal()` is Orbix specific. The memory returned by `get_principal` must be freed by the user.

**See Also** `CORBA::Principal`

**CORBA::BOA::getFileDescriptors()**

- Synopsis** `#include <sys/types.h>  
fd_set CORBA::BOA::getFileDescriptors() const;`
- Description** Gets the set of file descriptors scanned by Orbix to detect incoming events. When using libraries or systems that depend on the TCP/IP `select()` call you may need to know which file descriptors are scanned by Orbix.
- Return Value** Returns a set of file descriptors.
- Notes** Orbix specific.
- See Also** `CORBA::Object::_fd()`

**CORBA::BOA::getFilter()**

- Synopsis** `CORBA::Filter_ptr CORBA::BOA::getFilter();`
- Description** Gets the first per-process filter (if any) in the chain of filters associated with the server process.
- Return Value** Returns a pointer to the first filter object in the chain, if any.
- Notes** Orbix specific.
- See Also** `CORBA::Filter`

### CORBA::BOA::impl\_is\_ready()

#### Synopsis

```
void CORBA::BOA::impl_is_ready(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
void CORBA::BOA::impl_is_ready(ImplementationDef_ptr serverName,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
void CORBA::BOA::impl_is_ready(
 CORBA::ImplementationDef_ptr serverName = "",
 CORBA::ULong timeOut = CORBA::Orbix.DEFAULT_TIMEOUT,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

#### Description

Once a server is registered with Orbix, a process is automatically launched to run it if an operation is invoked on one of its objects. Once launched, the server should initialize itself, creating any objects it requires, and it should then call `CORBA::Orbix.impl_is_ready()` to indicate that it has completed its initialization and is ready to receive operation requests on its objects.

The `impl_is_ready()` function normally does not return immediately; it blocks the server until an event occurs, handles the event, and re-blocks the server to await another event. (The functions `CORBA::BOA::ProcessEvents()` and `CORBA::BOA::ProcessNextEvent` provide alternative ways of handling events.)

The `impl_is_ready()` function returns only when:

- ◆ A time-out occurs. A server can time out either because it has no clients for the timeout duration, or because none of its clients use it for that period.
- ◆ An exception occurs while waiting for or processing an event.
- ◆ The function `CORBA::BOA::deactivate_impl()` is called.

A persistent server (one that is run manually rather than being launched by Orbix) should call the `impl_is_ready()` function *before* it has any interaction with Orbix. For example, a persistent server should not pass out an object reference for one of its objects (for example, as a parameter or return value, or even by printing it) until `impl_is_ready()` is called. Such an object reference would not have the correct server name since Orbix has no way of determining this before `impl_is_ready()` is called.

---

**Note:** The implementation of `impl_is_ready()` inserts the correct server name into the object names of the server's objects, but it cannot do so for any object references that have already been passed out of the address space.

---



Other interactions with Orbix, such as calling an operation on a remote object, also cause difficulties if they occur in a persistent server before `impl_is_ready()` is called. You can circumvent this problem by calling the `CORBA::ORB::setServerName()` function on the `CORBA::Orbix` object before making external calls.

Persistent servers, once they have called `impl_is_ready()` behave as shared activation mode servers. However, if a server is registered as unshared or per-method, then, as required by the CORBA specification, `impl_is_ready()` fails if the server is launched manually.

Normally, a server must be registered in the Implementation Repository (using `putit` or the Orbix Server Manager GUI utility) before it can call `impl_is_ready()`. However, if the `-u` switch is specified to the `orbixd` daemon, a persistent server can call `impl_is_ready()` without being registered.

### Parameters

`server_name`     The `server_name` parameter is optional if the server is launched by Orbix; it is required if the server is launched manually or externally to Orbix (that is, for a CORBA persistent server). It is recommended, therefore, that you specify the `server_name` parameter. If you specify the `server_name` parameter, it must be exactly the server name with which the server was registered. The `server_name` parameter need not be the name of a class or interface; it is the name of a server, registered with the Implementation Repository. If you do not wish to specify the `server_name`, but wish to specify a non-default `timeOut` (or `Environment`), you should pass a zero length string ("") as the value of the `name` parameter.

`timeOut` Indicates the number of milliseconds to wait between events; a time-out occurs if Orbix has to wait longer than the specified timeout for the next event. A time-out of zero indicates that `impl_is_ready()` should time out and return immediately *without* checking if there is any pending event. A time-out does not cause `impl_is_ready()` to raise an exception. You can pass the default time-out explicitly as `CORBA::Orbix.DEFAULT_TIMEOUT`. You can specify an infinite time-out by passing `CORBA::Orbix.INFINITE_TIMEOUT`. The `timeOut` parameter is meaningless for the per-method call activation mode, since the process terminates once the operation call that caused it to be launched has completed.

**Notes** The member function `CORBA::BOA::impl_is_ready(ImplementationDef server_name)` is CORBA compliant. The overloaded alternatives are Orbix specific.

**See Also** `CORBA::BOA::deactivate_impl()`  
`CORBA::BOA::obj_is_ready()`  
`CORBA::BOA::processEvents()`  
`CORBA::BOA::processNextEvent()`  
`CORBA::ORB::setServerName()`

### **CORBA::BOA::isDeactivated()**

**Synopsis** `CORBA::Boolean CORBA::BOA::isDeactivated() const;`

**Description** Returns the current setting of the `isDeactivated` flag.

#### **Return Value**

0 (FALSE) The server wants to continue to receive requests; that is the function calls `CORBA::BOA::impl_is_ready()`, `CORBA::BOA::obj_is_ready()`, and `CORBA::BOA::processEvents()` do not exit after receipt of the next incoming event (operation calls, connections, disconnections). This is the default behavior.

1 (TRUE) The server wants to stop receiving requests.

**Notes** Orbix specific.

**See Also** CORBA::BOA::deactivate\_impl()  
CORBA::BOA::deactivate\_obj()

### **CORBA::BOA::isEventPending()**

**Synopsis** CORBA::Boolean CORBA::BOA::isEventPending(  
CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv())  
const;

**Description** Tests whether or not there is an outstanding event, that is whether or not CORBA::BOA::processNextEvent() would block the server for a period.

**Return Value** Returns 1 (TRUE) if there is a pending event, returns 0 (FALSE) otherwise.

**Notes** Orbix specific.

**See Also** CORBA::BOA::processNextEvent()

### **CORBA::BOA::myActivationMode()**

**Synopsis** CORBA::BOA::activationMode CORBA::BOA::myActivationMode(  
CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());

**Description** Determines the fundamental activation mode with which the server was launched: shared, unshared, persistent, or per-method.

**Return Value** Returns the activation mode.

**Exceptions** If called within a client application, it raises the CORBA::NO\_IMPLEMENT exception and returns unknownActivationMode.

**Notes** Orbix specific.

**See Also** CORBA::BOA::activationMode

### **CORBA::BOA::myImplementationName()**

**Synopsis** const char\* CORBA::BOA::myImplementationName(  
CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv())  
const;

**Description** Finds the server's name as registered in the Implementation Repository. For a persistent server, the contents of the string are unspecified until `CORBA::BOA::impl_is_ready()`, `CORBA::BOA::obj_is_ready()` or `CORBA::ORB::setServerName()` is called.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::impl_is_ready()`  
`CORBA::ORB::setServerName()`

### **CORBA::BOA::myImpRepPath()**

**Synopsis**

```
const char* CORBA::BOA::myImpRepPath(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()
 const;
```

**Description** Finds the path name of the Implementation Repository directory in which the server is registered.

**Notes** Orbix specific.

### **CORBA::BOA::myIntRepPath()**

**Synopsis**

```
const char* CORBA::BOA::myIntRepPath(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()
 const;
```

**Description** Finds the name of the directory used to store information about the interfaces in the Interface Repository. This directory contains the appropriate information if the `-R` switch was passed to the IDL compiler.

**Notes** Orbix specific.

### **CORBA::BOA::myMarkerName()**

**Synopsis**

```
const char* CORBA::BOA::myMarkerName(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()
 const;
```

- Description** Finds the marker name of the activation object that caused the server to be launched. For a persistent or a per-method server, this marker name is "\*".
- Notes** Orbix specific.
- See Also** CORBA::BOA::myMarkerPattern()

### **CORBA::BOA::myMarkerPattern()**

- Synopsis**
- ```
const char* CORBA::BOA::myMarkerPattern(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```
- Description** Finds the marker pattern that the activation object matched in the Implementation Repository and hence caused this server to be launched.
- For a persistent or per-method server, this pattern is "*".
- Marker patterns are explained in the *Orbix C++ Administrator's Guide*.
- Notes** Orbix specific.
- See Also** CORBA::BOA::myMarkerName()

CORBA::BOA::myMethodName()

- Synopsis**
- ```
const char* CORBA::BOA::myMethodName(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;
```
- Description** Finds the method that caused server to be launched. For a non per-method server, this value is null.
- Notes** Orbix specific.

### **CORBA::BOA::obj\_is\_ready()**

- Synopsis**
- ```
void CORBA::BOA::obj_is_ready(Object_ptr obj,  
    CORBA::ImplementationDef_ptr impl,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());  
void CORBA::BOA::obj_is_ready(Object_ptr obj,  
    CORBA::ImplementationDef_ptr impl,
```

```
CORBA::ULong timeOut = CORBA::Orbix.DEFAULT_TIMEOUT,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description A server running in the unshared activation mode (that is, with one registered object per process) may call the `CORBA::BOA::obj_is_ready()` function on the `CORBA::Orbix` object to indicate that it has completed its initialization. The server remains active and will receive requests for its registered object until:

- ◆ It calls `CORBA::BOA::deactivate_obj()`.
- ◆ The call to `obj_is_ready()` times out.
- ◆ An exception is raised.

Parameters

`obj` The registered object that the process manages.

`impl` The server name.

`timeOut` The time-out period. A server can time out either because it has no clients for the time-out duration, or because none of its clients use it for that period. The default time-out is given by `CORBA::Orbix::DEFAULT_TIMEOUT`. An infinite time-out is specified by `CORBA::Orbix::INFINITE_TIMEOUT`.

Notes `CORBA::BOA::obj_is_ready(ObjectRef obj, ImplementationDef impl)` is CORBA compliant. The overloaded alternative is Orbix specific.

See Also `CORBA::BOA::deactivate_obj()`
`CORBA::BOA::impl_is_ready()`

CORBA::BOA::processEvents()

Synopsis

```
Status CORBA::BOA::processEvents(CORBA::ULong timeOut =  
CORBA::Orbix.DEFAULT_TIMEOUT,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description There are three kinds of Orbix events:

- ◆ Operation requests
- ◆ Connections from clients
- ◆ Disconnections of clients

If a zero time-out period is given to `CORBA::BOA::impl_is_ready()` or `CORBA::BOA::obj_is_ready()`, when invoked on the `CORBA::Orbix` object, the call returns immediately—allowing a program to subsequently state at what points it is willing to accept incoming Orbix events.

The function `processEvents()` blocks the server until an event arrives, handles the event, and continues to process events, until none arrives within the time-out period. It has the same effect as calling `CORBA::BOA::processNextEvent()` repeatedly until it times out.

The function `processEvents()` is similar in functionality to `impl_is_ready()` (or `obj_is_ready()`) because it processes any number of events until it times out. However, use of `processEvents()` does not initialize the server and therefore does not fulfil a server's requirement to call `impl_is_ready()` (or `obj_is_ready()`).

One example of using `processEvents()` is where a manually-launched server wishes to interact with Orbix (for example, by calling a remote operation or by passing out or printing an object reference for one of its objects) before it is ready to handle events. Before it interacts with Orbix, it must call `impl_is_ready()` (or `obj_is_ready()`), in this case with a zero time-out, and then call `processEvents()` when it is ready to handle events. Alternatively, before it interacts with Orbix it may call `CORBA::ORB::setServerName()` and then call `processEvents()`.

Parameters

`timeOut` Indicates how long (in milliseconds) the server should be blocked. A time-out of zero indicates that `processEvents()` should not block; it returns immediately if there is no waiting event.

Return Value Normally returns 0 (FALSE). Returns 1 (TRUE) if an exception occurs while waiting for or processing an event or if the ORB has been deactivated.

Notes Orbix specific.

See Also `CORBA::BOA::processNextEvent()`
`CORBA::BOA::impl_is_ready()`
`CORBA::BOA::obj_is_ready()`
`CORBA::ORB::setServerName()`

CORBA::BOA::processNextEvent()

Synopsis

```
Status CORBA::BOA::processNextEvent(CORBA::ULong timeOut =
    CORBA::Orbix.DEFAULT_TIMEOUT,
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description

You may wish to have more control over the handling of events in a server. There are three kinds of events:

- ◆ Operation requests
- ◆ Connections from clients
- ◆ Disconnections of clients

This function blocks the server until an event arrives, handles that one event, and normally returns zero.

If a zero time-out period is given to `CORBA::BOA::impl_is_ready()` or `CORBA::BOA::obj_is_ready()`, the call returns immediately—allowing a program to subsequently state at what points it is willing to accept incoming Orbix events. You can do this by calling `CORBA::Orbix.processNextEvent()`.

Parameters

`timeOut` Indicates how long (in milliseconds) the server should be blocked. A time-out of zero indicates that `processNextEvent()` should not block; it returns immediately if there is no waiting event.

Return Value

Normally returns 0 (`FALSE`). Returns 1 (`TRUE`) if an exception occurs while waiting for or processing an event or if the ORB has been deactivated.

Notes

Orbix specific.

See Also

```
CORBA::BOA::processEvents()
CORBA::BOA::impl_is_ready()
CORBA::BOA::obj_is_ready()
CORBA::BOA::deactivate_impl()
```

CORBA::BOA::propagateTIEdelete()

Synopsis

```
CORBA::Boolean CORBA::BOA::propagateTIEdelete(Boolean value,
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description By default, deletion of a TIE (calling `CORBA::release()` on a TIE with a reference count of one) results in the deletion of the implementation object pointed to by the TIE.

Normally this is the required behaviour, but if not, you should call `propagateTIEdelete(FALSE)` on the `CORBA::Orbix` object to ensure that the implementation object is *never* deleted by Orbix.

You can specify more than one TIE for the same implementation object. When any of these TIEs is deleted, the implementation object itself is, by default, deleted. You may wish to call `propagateTIEdelete(FALSE)` to ensure that this does not happen.

Note: If you are using multiple TIEs to a single object, when `propagateTIEdelete(TRUE)` is in force, you should be aware that deletion of any one of these TIEs leaves the other TIEs dangling.

Return Value This function returns the previous setting. The default setting is true; that is, the implementation object is deleted when the TIE object is deleted.

Notes Orbix specific.

CORBA::BOA::propagateTIEdelete()

Synopsis `CORBA::Boolean CORBA::BOA::propagateTIEdelete() const;`

Description Returns the current value of the `propagateTIEdel` flag.

Return Value

1 (TRUE)	Deletion of a TIE object results in the deletion of the implementation object pointed to by the TIE. This is the default behavior.
0 (FALSE)	The implementation object is <i>never</i> deleted by Orbix when the TIE object that points to the implementation object is deleted.

Note that when using multiple TIE objects pointing to a single implementation object, if `propagateTIEdelete` is `TRUE`, then deleting any of the TIE objects leaves the other TIE objects with dangling references.

Notes Orbix specific.

See Also `CORBA::BOA::propagateTIEdelete(CORBA::Boolean)`

CORBA::BOA::setImpl()

Synopsis

```
static void CORBA::BOA::setImpl(const char* InterfaceName,  
    CORBA::DynamicImplementation& DSISkeleton,  
    const char* MarkerServer = "",  
    CORBA::LoaderClass* Loader = 0 );
```

Description

Registers an instance of `CORBA::DynamicImplementation` to handle requests of a given interface.

Parameters

<code>InterfaceName</code>	The fully-scoped name of an interface that this <code>DynamicImplementation</code> object handles.
<code>DSISkeleton</code>	An instance of <code>CORBA::DynamicImplementation</code> that handles requests for the interface specified in <code>InterfaceName</code> .
<code>MarkerServer</code>	Allows an object (or set of objects) to be specified such that this <code>DynamicImplementation</code> object handles only that object (or set of objects). The parameter has the format <code>marker:server</code> . If you specify no server, the server name defaults to the name specified in <code>InterfaceName</code> . If you specify no marker, this <code>DynamicImplementation</code> object handles all objects within the specified (or defaulted) server.
<code>Loader</code>	A pointer to a loader for instances of the interface specified in <code>InterfaceName</code> .

Notes CORBA compliant.

See Also `CORBA::DynamicImplementation`

CORBA::BOA::setNoHangup()

Synopsis

```
CORBA::Boolean CORBA::BOA::setNoHangup(CORBA::Boolean);
```

Description

When calling the functions `CORBA::BOA::impl_is_ready()`, `CORBA::BOA::obj_is_ready()`, and `CORBA::BOA::processEvents()`, the user may specify the amount of time the server remains waiting for incoming events. Events may be operation calls, connections, or disconnections. If there are no events received by the server within this time-out period then these functions will return. This is true even if clients are connected to the server.

A server may prefer to remain active while there are clients connected, active or not, by calling `setNoHangup(TRUE)`. The time-out period is sixty seconds by default.

Parameters

value 0 (FALSE) - The functions `CORBA::BOA::impl_is_ready()`, `CORBA::BOA::obj_is_ready()`, and `CORBA::BOA::processEvents()` always timeout and return after the time-out period has expired even if clients are connected. This is the default behavior.

1 (TRUE) - The time-out period specified when calling the functions `CORBA::BOA::impl_is_ready()`, `CORBA::BOA::obj_is_ready()`, and `CORBA::BOA::processEvents()` specifies the amount of time a server remains waiting when there are no client connections to it. If there are connected clients, then these functions will not return.

Return Value Returns the previous value of the `noHangup` flag for the server.

Notes Orbix specific.

See Also `CORBA::BOA::getNoHangup()`

CORBA::BOA::usingLoaders()

Synopsis

```
CORBA::Boolean CORBA::BOA::usingLoaders() const;
```

Description

Returns the current value of the `useLoaders` flag.

Return Value

1 (TRUE)	The use of loaders is enabled. This is the default value.
0 (FALSE)	The use of loaders is disabled.

Notes Orbix specific.

See Also `CORBA::BOA::enableLoaders (CORBA::Boolean)`
`CORBA::LoaderClass`

CORBA::Context

Synopsis

Class `CORBA::Context` implements the OMG pseudo-interface `Context`. A context is intended to represent information about the client that is inconvenient to pass via parameters.

An IDL operation can specify that it is to be provided with the client's mapping for particular identifiers (properties)—it does this by listing these identifiers following the operation declaration in a `context` clause. An IDL operation that specifies a `context` clause is mapped to a C++ member function that takes an extra input parameter of type `Context_ptr`, just before the `Environment` parameter. A client can optionally maintain one or more CORBA `Context` objects, which provide a mapping from identifiers (string names) to string values. A `Context` object contains a list of properties; each property consists of a name and a string value associated with that name and can be passed to a function that takes a `Context` parameter.

You can arrange `Contexts` in a hierarchy by specifying parent-child relationships among them. Then, a child context passed to an operation also includes the identifiers of its parent and all subsequent parent contexts.

CORBA

```
// Pseudo IDL
pseudo interface Context {
    readonly attribute Identifier context_name;
    readonly attribute Context parent;

    Status create_child(
        in Identifier child_ctx_name, out Context child_ctx);

    Status set_one value(
        in Identifier propname, in any propvalue);
    Status set_values(in NVList values);
    Status delete_values(in Identifier propname);
    Status get_values(in Identifier start_scope,
        in Flags op_flags,
        in Identifier pattern,
        out NVList values);
};
```

Orbix

```
// C++
class Context {
public:
    Status set_one_value(const char* prop_name,
                        const Any& value,
                        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

    Status set_values(NVList_ptr values,
                     CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

    Status get_values(const char* start_scope,
                     Flags op_flags,
                     const char* prop_name,
                     NVList_ptr& values,
                     CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

    Status delete_values(const char* prop_name,
                        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

    Status create_child(const char* ctx_name,
                       Context_ptr& child_ctx,
                       CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

    const char* context_name(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
    const;

    Context_ptr parent(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
    const;

    ORBStatus _delete (const CORBA::Flags& del_flags,
                      CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

    ULong get_count(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
    const;

    ULong get_count_all(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
    const;
```

```
Context(Context* parent = 0);

Context (const char* name, Context* parent=0);

~Context();

void encode (Request&, char*) const;

Context(Request&);

static Context_ptr __stdcall IT_create(
    Context_ptr parent = 0,
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

static Context_ptr __stdcall IT_create(
    const char* name,
    Context_ptr parent = 0,
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

static Context_ptr __stdcall IT_create(
    Request& IT_r,
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

static Context_ptr __stdcall _duplicate(
    Context_ptr obj,
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

static Context_ptr __stdcall _nil(
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
};
```

Notes CORBA compliant.

See Also CORBA::ContextIterator
CORBA::NVList
CORBA::Flags

CORBA::Context::Context()

Synopsis `CORBA::Context::Context(Context* parent = 0);`

Description Creates a new context (possibly a child context).

Parameters

`parent` The parent context, if any.

Notes Orbix specific. Refer to `CORBA::Context::create_child()` for a CORBA-compliant function to create a child `Context`.

See Also `CORBA::Context::create_child()`
`CORBA::Context::IT_create()`
Other `Context` constructors.

CORBA::Context::Context()

Synopsis `CORBA::Context::Context(const char* name, Context* parent = 0);`

Description Creates a new context (possibly a child context) with the given context name.

Parameters

`name` The name of the context.

`parent` The parent context, if any.

Notes Orbix specific. Refer to `CORBA::Context::create_child()` for CORBA-compliant function to create a child `Context`.

See Also `CORBA::Context::create_child()`
`CORBA::Context::IT_create()`
Other `Context` constructors.

CORBA::Context::Context()

- Synopsis** `CORBA::Context::Context (Request&)`
- Description** Conversion from a Request.
- Notes** Orbix specific.
- See Also** `CORBA::Context::create_child()`
`CORBA::Context::IT_create()`
Other Context constructors.

CORBA::Context::~~Context()

- Synopsis** `CORBA::Context::~~Context();`
- Description** Destructor.
- Notes** Orbix specific. Calling `CORBA::release()` on the Context is the CORBA compliant way to free a Context created using `IT_create()` or `create_child()`. The `release()` function frees child contexts.
- See Also** `CORBA::release()`

CORBA::Context::_duplicate()

- Synopsis** `static CORBA::Context::Context_ptr CORBA::Context::_duplicate(
Context_ptr obj,
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Increments the reference count of obj.
- Return Value** Returns a reference to self.
- Notes** CORBA compliant.
- See Also** `CORBA::release()`

CORBA::Context::_nil()

Synopsis	<pre>static CORBA::Context_ptr CORBA::Context::_nil(CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());</pre>
Description	Returns a nil object reference for a Context object.
Notes	CORBA compliant.
See Also	<code>CORBA::is_nil()</code>

CORBA::Context::context_name()

Synopsis	<pre>const char* CORBA::Context::context_name(CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) const;</pre>
Description	Returns the name of the Context object.
See Also	<code>CORBA::Context::create_child()</code>

CORBA::Context::create_child()

Synopsis	<pre>Status CORBA::Context::create_child(const char* ctx_name, Context_ptr& child_ctx, CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());</pre>				
Description	Creates a child context of the current context. When a child context is passed as a parameter to an operation, any searches (using <code>CORBA::Context::get_values()</code>) looks in parent contexts if necessary to find matching property names.				
Parameters	<table><tr><td><code>ctx_name</code></td><td>The name of the child context. Context object names follow the rules for IDL identifiers.</td></tr><tr><td><code>child_ctx</code></td><td>The newly created context.</td></tr></table>	<code>ctx_name</code>	The name of the child context. Context object names follow the rules for IDL identifiers.	<code>child_ctx</code>	The newly created context.
<code>ctx_name</code>	The name of the child context. Context object names follow the rules for IDL identifiers.				
<code>child_ctx</code>	The newly created context.				
Return Value	Returns 1 (TRUE) if successful; returns 0 (FALSE) otherwise.				
Notes	CORBA compliant.				
See Also	<code>CORBA::Context::get_values()</code>				

CORBA::Context::delete_values()

- Synopsis** `Status CORBA::Context::delete_values(const char* prop_name,
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Deletes the specified property value(s) from the context. The search scope is limited to the Context object on which the invocation is made.
- Parameters**
- `prop_name` The property name to be deleted. If `prop_name` has a trailing '*', all matching properties are deleted.
- Return Value** Returns 1 (TRUE) if successful; returns 0 (FALSE) otherwise. An exception is raised if no matching property is found.
- Notes** CORBA compliant.

CORBA::Context::get_count()

- Synopsis** `CORBA::Long CORBA::Context::get_count(
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;`
- Description** Finds the number of property/value pairs in the context.
- Return Value** The number of property/value pairs in the supplied context.
- Notes** Orbix specific.
- See Also** `CORBA::Context::get_count_all()`

CORBA::Context::get_count_all()

- Synopsis** `CORBA::Long CORBA::Context::get_count_all(
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;`
- Description** Finds the number of property/value pairs in this context, its parent contexts, and all subsequent parent contexts.
- Return Value** The total number of property/value pairs in the supplied context, its parent context (if any), and all subsequent parent contexts (if any).

Notes Orbix specific.

See Also `CORBA::Context::get_count()`

CORBA::Context::get_values()

Synopsis

```
Status CORBA::Context::get_values(  
    const char* start_scope,  
    const Flags op_flags,  
    const char* prop_name,  
    CORBA::NVList_ptr& values,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description Retrieves the specified context property values.

Parameters

<code>start_scope</code>	The context in which the search for the values requested should be started. The name of a direct or indirect parent context may be specified to this parameter. If 0 is passed, the search begins in the context which is the target of the call.
<code>op_flags</code>	By default, searching of identifiers propagates upwards to parent contexts; if <code>CORBA::CTX_RESTRICT_SCOPE</code> is specified, then searching is limited to the specified search scope or context object.
<code>prop_name</code>	If <code>prop_name</code> has a trailing '*', all matching properties and their values are returned.
<code>values</code>	An <code>NVList</code> to contain the returned property values.

Return Value Returns 1 (`TRUE`) if matching properties are found; returns 0 (`FALSE`) otherwise.

Notes CORBA compliant.

CORBA::Context::IT_create()**Synopsis**

```
static CORBA::Context_ptr CORBA::Context::IT_create(  
    CORBA::Context_ptr parent=0,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());  
static CORBA::Context_ptr CORBA::Context::IT_create(  
    const char* name,  
    CORBA::Context_ptr parent=0,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());  
static CORBA::Context_ptr CORBA::Context::IT_create(  
    CORBA::Request& IT_r,  
    CORBA::Environment& env = CORBA::IT_chooseDefaultEnv());
```

Description

In the absence of a CORBA-specified way to create a (top level) Context pseudo object in the current standard C++ mapping, Orbix provides the `IT_create()` function to initialize an object reference for a Context.

Use of this function is recommended in preference to C++ operator `new` to ensure portability across future Orbix versions and for memory management consistency.

Notes

Orbix specific. Refer to `CORBA::Context::create_child()` for the CORBA compliant way to create a child Context.

See Also

`CORBA::Context::create_child()`
Context constructors.

CORBA::Context::parent()**Synopsis**

```
CORBA::Context_ptr CORBA::Context::parent(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())  
const;
```

Description

Returns the parent of the Context object.

Notes

CORBA compliant.

See Also

`CORBA::Context::create_child()`

CORBA::Context::set_one_value()

- Synopsis** `Status CORBA::Context::set_one_value(const char* prop_name,
 const CORBA::Any& value,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Adds property name and value to context. Although the `value` member is of type `Any`, the type of the `Any` must be a string.
- Parameters**
- | | |
|------------------------|-----------------------------------|
| <code>prop_name</code> | The name of the property to add. |
| <code>value</code> | The value of the property to add. |
- Return Value** Returns 1 (`TRUE`) if successful; returns 0 (`FALSE`) otherwise.
- Notes** CORBA compliant.
- See Also** `CORBA::Context::set_values()`

CORBA::Context::set_values()

- Synopsis** `Status CORBA::Context::set_values(const CORBA::NVList_ptr values,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Sets one or more property values in the context. The previous value of the property, if any, is discarded.
- Parameters**
- | | |
|---------------------|--|
| <code>values</code> | An <code>NVList</code> containing the <code>property_name:values</code> to add or change. In the <code>NVList</code> , the <code>flags</code> field must be set to zero, and the <code>TypeCode</code> associated with an attribute value must be <code>CORBA::_tc_string</code> . |
|---------------------|--|
- Return Value** Returns 1 (`TRUE`) if successful; returns 0 (`FALSE`) otherwise.
- Notes** CORBA compliant.
- See Also** `CORBA::Context::set_one_value()`

CORBA::ContextIterator

Synopsis Class CORBA::ContextIterator defines a C++ iterator class for CORBA::Context.

Orbix

```
// C++
class ContextIterator {
public:
    ContextIterator(const Context*);
    ~ContextIterator();

    char* operator()();
};
```

Notes Orbix specific.

See Also CORBA::Context

CORBA::ContextIterator::ContextIterator()

Synopsis CORBA::ContextIterator::ContextIterator(
const CORBA::Context* context);

Description Constructor. Creates an iterator for context.

Notes Orbix specific.

CORBA::ContextIterator::~~ContextIterator()

Synopsis CORBA::ContextIterator::~~ContextIterator();

Description Destructor.

Notes Orbix specific.

CORBA::ContextIterator::operator()

Synopsis

```
char* operator() ();
```

Description

The *i*th call, where *i* is even, returns the name of a property in the `Context`. Where *i* is odd, the *i*th call returns the value of a property in the `Context` whose name is that returned by the (*i*-1)th call. Remember that within a `Context` object the information is stored in property name/value pairs.

Notes

Orbix specific.

CORBA::DynamicImplementation

Synopsis A server that uses the Dynamic Skeleton Interface (DSI) must create one or more objects that support the IDL interface `CORBA::DynamicImplementation` and register these objects with Orbix using `CORBA::BOA::setImpl()`.

CORBA

```
// IDL
pseudo interface DynamicImplementation {
    void invoke(inout ServerRequest request,
               inout Environment env);
```

Orbix

```
// C++
class DynamicImplementation {
public:
    virtual void invoke(ServerRequest& request,
                       Environment& env,
                       CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) =
0;
protected:
    DynamicImplementation();
    virtual ~DynamicImplementation();
};
```

Notes CORBA compliant.

See Also `CORBA::ServerRequest`
`CORBA::BOA::setImpl()`

CORBA::DynamicImplementation:: DynamicImplementation()

Synopsis `CORBA::DynamicImplementation::DynamicImplementation();`

Description Default constructor.

Notes CORBA compliant.

CORBA::DynamicImplementation:: ~DynamicImplementation()

Synopsis	<code>DynamicImplementation::~DynamicImplementation();</code>
Description	Destructor.
Notes	CORBA compliant.

CORBA::DynamicImplementation::invoke()

Synopsis	<pre>virtual void DynamicImplementation::invoke(S CORBA::ServerRequest& request, CORBA::Environment& env, CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) = 0;</pre>				
Description	The <code>invoke()</code> function is informed of incoming operation and attribute requests to a server. An implementation of <code>invoke()</code> (in a derived class of <code>CORBA::DynamicImplementation</code>) is known as a Dynamic Implementation Routine (DIR).				
Parameters	<table><tr><td><code>request</code></td><td>Contains details of the request to be invoked. This object is created by Orbix when it receives an incoming request and recognises it as one to be handled by the DSI: that is, an instance of <code>DynamicImplementation</code> has been registered to handle the target interface.</td></tr><tr><td><code>env</code></td><td>Contains the environment associated with the <code>request</code> parameter.</td></tr></table>	<code>request</code>	Contains details of the request to be invoked. This object is created by Orbix when it receives an incoming request and recognises it as one to be handled by the DSI: that is, an instance of <code>DynamicImplementation</code> has been registered to handle the target interface.	<code>env</code>	Contains the environment associated with the <code>request</code> parameter.
<code>request</code>	Contains details of the request to be invoked. This object is created by Orbix when it receives an incoming request and recognises it as one to be handled by the DSI: that is, an instance of <code>DynamicImplementation</code> has been registered to handle the target interface.				
<code>env</code>	Contains the environment associated with the <code>request</code> parameter.				
Notes	CORBA compliant.				
See Also	<code>CORBA::ServerRequest</code>				

CORBA::Environment

Synopsis

Class `CORBA::Environment` implements the OMG pseudo-interface `Environment`. You can use it to pass information between a client and a server. You can use this default parameter, in Orbix, to set a time-out value for remote calls and to pass security information. When the C++ host compiler supports C++ exception handling, use of the default `Environment` parameter for exception handling is not CORBA compliant (as prescribed in the CORBA 2.0 C++ mapping).

Note: Use of class `CORBA::Environment` for exception handling is deprecated, and may not be supported in future releases of Orbix.

CORBA

```
// Pseudo IDL
pseudo interface Environment {
    attribute exception exception;
    void clear();
};
```

Orbix

```
// C++
class Environment : public IT_PseudoIDL {
public:

    Request* request();
    void request(Request* r);

    Environment(Exception* exc);
    Environment(SystemException* se);
    Environment(Request& r);
    Environment();

    ~Environment();

    Environment(const Environment& IT_env);

    Environment& operator = (const Environment& IT_env);
    Environment& operator = (Exception* exc);
```

```
Environment& operator = (SystemException* se);

operator int() const;

void exception(Exception* exc);
Exception* exception() const;

void clear();

ULong timeout() const;
void timeout(ULong t);

static Environment_ptr IT_create(
    Exception* e,
    Environment& IT_env = IT_chooseDefaultEnv());

static Environment_ptr IT_create(
    Request& r,
    Environment& IT_env = IT_chooseDefaultEnv());

static Environment_ptr IT_create(
    Environment& IT_env = IT_chooseDefaultEnv());

static Environment_ptr IT_create(
    const Environment&,
    Environment& IT_env = IT_chooseDefaultEnv());

static Environment_ptr IT_create(
    SystemException* se,
    Environment& IT_env = IT_chooseDefaultEnv());

static Environment_ptr _duplicate(
    Environment_ptr obj,
    Environment& IT_env = IT_chooseDefaultEnv());

static Environment_ptr _nil(
    Environment& IT_env = IT_chooseDefaultEnv());
};
```

Notes CORBA compliant.

See Also CORBA::IT_chooseDefaultEnv()
CORBA::BOA::get_principal()

CORBA::Environment::clear()

- Synopsis** `void CORBA::Environment::clear();`
- Description** Deletes the `Exception`, if any, contained in the `Environment`. This is equivalent to passing zero to `CORBA::Environment::exception(CORBA::Exception*)`. It is not an error to call `clear()` on an `Environment` that holds no exception.
- Notes** CORBA compliant.
- See Also** `CORBA::Environment::exception(Exception*)`

CORBA::Environment::_duplicate()

- Synopsis**

```
static CORBA::Environment_ptr CORBA::Environment::_duplicate(
    CORBA::Environment_ptr obj,
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```
- Description** Increments the reference count of `obj`.
- Return Value** Returns a reference to itself.
- Notes** CORBA compliant.
- See Also** `CORBA::release()`

CORBA::Environment::Environment()

- Synopsis** `CORBA::Environment::Environment();`
- Description** Default constructor.
- Notes** Orbix specific. Use of this constructor is compliant when the C++ environment does not support C++ exception handling. Refer to `CORBA::ORB::create_environment()` for the CORBA compliant way to create an `Environment` when the C++ environment supports exception handling.
- See Also** `CORBA::ORB::create_environment()`
`CORBA::Environment::IT_create()`
Other `Environment` constructors.

CORBA::Environment::Environment()

Synopsis	<code>CORBA::Environment::Environment (CORBA::Exception* e);</code>
Description	Conversion from an <code>Exception</code> . Constructs an <code>Environment</code> that contains the exception denoted by <code>e</code> .
Notes	Orbix specific.
See Also	<code>CORBA::Exception</code> <code>CORBA::ORB::create_environment()</code> <code>CORBA::Environment::IT_create()</code> Other <code>Environment</code> constructors.

CORBA::Environment::Environment()

Synopsis	<code>CORBA::Environment::Environment (const CORBA::Environment& env);</code>
Description	Copy constructor.
Notes	Orbix specific.
See Also	<code>CORBA::ORB::create_environment()</code> <code>CORBA::Environment::IT_create()</code> Other <code>Environment</code> constructors.

CORBA::Environment::Environment()

Synopsis	<code>CORBA::Environment::Environment (CORBA::SystemException* se);</code>
Description	Conversion from a <code>SystemException</code> . Constructs an <code>Environment</code> that contains the exception denoted by <code>se</code> .
Notes	Orbix specific. Use of <code>CORBA::Environment::</code> for exception handling is not CORBA-compliant. Therefore the use of this API is deprecated.
See Also	<code>CORBA::SystemException</code> <code>CORBA::ORB::create_environment()</code> <code>CORBA::Environment::IT_create()</code> Other <code>Environment</code> constructors.

CORBA::Environment::~~Environment()

Synopsis	<code>CORBA::Environment::~~Environment();</code>
Description	Destructor.
Notes	Orbix specific.

CORBA::Environment::exception()

Synopsis	<code>CORBA::Exception* CORBA::Environment::exception() const;</code>
Description	Returns the exception, if any, raised by a preceding remote request. For example:

```
// C++
CORBA::Environment env;
A_var obj = ...
obj->op(env);
if(CORBA::Exception* ex = env.exception()) {
    ...
}
```

You can make a number of remote requests using the same `Environment` variable. Each attempt at a request immediately aborts if the `Exception` referenced by the `Environment` is not 0, and thus any failure causes subsequent requests not to be attempted, until the exception pointer is reset to 0. Any failed call may also generate one or more null proxies, so that any attempts to use these proxies prior to the end of an Orbix `TRY` macro (for non exception-handling compilers) are null operations.

Return Value	The <code>Environment</code> retains ownership of the <code>Exception</code> returned. Thus, once the <code>Environment</code> is destroyed, or its <code>Exception</code> cleared, the reference is no longer valid.
Notes	CORBA compliant.
See Also	<code>CORBA::Environment::exception()</code> <code>CORBA::Environment::exception(CORBA::Exception* e)</code> <code>CORBA::Environment::clear()</code>

CORBA::Environment::exception()

- Synopsis** `void CORBA::Environment::exception(CORBA::Exception* e)`
- Description** Assigns the `Exception` denoted by `e` into the `Environment`. The `Environment` assumes ownership of `e`; it does not copy `e`. The exception `e` must have been dynamically allocated.
- Notes** CORBA compliant.
- See Also** `CORBA::Environment::exception()`

CORBA::Environment::int()

- Synopsis** `operator CORBA::Environment::int() const;`
- Description** A conversion operator to convert an `Environment` to an `int`. It allows `Environment` objects to be used in conditions of statements such as `if` and `while`—typically by the Orbix exception macros for non-exception handling compilers.
- Notes** Orbix specific.

CORBA::Environment::IT_create()

- Synopsis**
- ```
static Environment_ptr CORBA::Environment::IT_create(
 CORBA::Exception* e,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
static Environment_ptr CORBA::Environment::IT_create(
 CORBA::Request& r,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
static Environment_ptr CORBA::Environment::IT_create(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
static Environment_ptr CORBA::Environment::IT_create(
 const CORBA::Environment&,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
static Environment_ptr CORBA::Environment::IT_create(
 CORBA::SystemException* se,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```



**Description** For consistency with other pseudo object types for which there is no CORBA specified way in the current standard C++ mapping to obtain an object reference, Orbix provides the `IT_create()` functions for class `Environment` to initialize an object reference. To ensure memory management consistency, you should not use the C++ `new` operator to create an `Environment`.

Refer to the corresponding constructors for details of the parameters to `IT_create()`.

**Notes** Orbix specific. See `CORBA::ORB::create_environment()` for the CORBA compliant way to create an `Environment`.

**See Also** `CORBA::ORB::create_environment()`  
`Environment` constructors.

### CORBA::Environment::\_nil()

**Synopsis**

```
static CORBA::Environment_ptr CORBA::Environment::_nil(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Returns a nil object reference for an `Environment` object.

**Notes** CORBA compliant.

**See Also** `CORBA::is_nil()`

### CORBA::Environment::operator=()

**Synopsis**

```
const CORBA::Environment& CORBA::Environment::operator = (
 const CORBA::Environment& env);
```

**Description** Assignment operator.

**Notes** Orbix specific.

**See Also** `CORBA::Environment::operator=(CORBA::Exception* e)`  
`CORBA::Environment::operator=(CORBA::SystemException* se)`

### CORBA::Environment::operator=()

**Synopsis**

```
const CORBA::Environment& CORBA::Environment::operator = (
```

```
CORBA::Exception* e);
```

- Description** Assignment from an Exception.
- Notes** Orbix specific. Use of `CORBA::Environment::` for exception handling is not CORBA-compliant. Therefore the use of this API is deprecated.
- See Also** `CORBA::Environment::operator=(const CORBA::Environment& env)`  
`CORBA::Environment::operator=(CORBA::SystemException* se)`

### **CORBA::Environment::operator=()**

- Synopsis**

```
const CORBA::Environment& CORBA::Environment::operator = (
 CORBA::SystemException* se);
```
- Description** Assignment from a System Exception.
- Notes** Orbix specific. Use of `CORBA::Environment::` for exception handling is not CORBA-compliant. Therefore the use of this API is deprecated.
- See Also** `CORBA::Environment::operator=(const Environment& env)`  
`CORBA::Environment::operator=(CORBA::Exception* e)`

### **CORBA::Environment::request()**

- Synopsis**

```
void CORBA::Environment::request(CORBA::Request* r);
```
- Description** Allows the user to associate a Request object with the Environment on which it is called.
- Parameters**
- `r` A pointer to the `CORBA::Request` object to be associated with the Environment on which it is called.
- Notes** Orbix specific.
- See Also** `CORBA::Environment::request()`

**CORBA::Environment::request()**

|                     |                                                                                                                                                                     |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>     | <code>CORBA::Request* CORBA::Environment::request();</code>                                                                                                         |
| <b>Description</b>  | Returns the Request object which has previously been associated with the Environment on which it is called.                                                         |
| <b>Return Value</b> | A pointer to the <code>CORBA::Request</code> object if any has been associated with the Environment on which it is called. Otherwise, the null pointer is returned. |
| <b>Notes</b>        | Orbix specific.                                                                                                                                                     |
| <b>See Also</b>     | <code>CORBA::Environment::request(CORBA::Request*)</code>                                                                                                           |

**CORBA::Environment::timeout()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                   |                |                                    |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|------------------------------------|
| <b>Synopsis</b>    | <code>void CORBA::Environment::timeout(CORBA::ULong t);</code>                                                                                                                                                                                                                                                                                                                    |                |                                    |
| <b>Description</b> | Sets the timeout for remote (non-oneway) calls for the Environment on which it is called. The value set by this function remains active until reset for the Environment. This timeout value supersedes any timeout set globally by <code>CORBA::ORB::defaultTxTimeout()</code> . This function is effective once a connection has been established between the client and server. |                |                                    |
| <b>Parameters</b>  | <table><tr><td><code>t</code></td><td>The timeout value in milliseconds.</td></tr></table>                                                                                                                                                                                                                                                                                        | <code>t</code> | The timeout value in milliseconds. |
| <code>t</code>     | The timeout value in milliseconds.                                                                                                                                                                                                                                                                                                                                                |                |                                    |
| <b>Exceptions</b>  | If a reply is not received within the given timeout interval, an invocation using this Environment value fails with a <code>CORBA::COMM_FAILURE</code> exception.                                                                                                                                                                                                                 |                |                                    |
| <b>Notes</b>       | Orbix specific.                                                                                                                                                                                                                                                                                                                                                                   |                |                                    |
| <b>See Also</b>    | <code>CORBA::Environment::timeout()</code><br><code>CORBA::ORB::defaultTxTimeout()</code>                                                                                                                                                                                                                                                                                         |                |                                    |

**CORBA::Environment::timeout()**

|                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <b>Synopsis</b>    | <code>CORBA::ULong CORBA::Environment::timeout() const;</code>      |
| <b>Description</b> | Gets the current timeout for the Environment on which it is called. |

**Return Value** The timeout, in milliseconds, for remote (non-oneway) calls for the Environment on which it is called.

**Notes** Orbix specific.

**See Also** `CORBA::Environment::timeout(CORBA::ULong)`  
`CORBA::ORB::defaultTxTimeout()`

# CORBA::Exception

**Synopsis** Class CORBA::Exception is a base class for all system and user-defined exception classes.

**Orbix**

```
// C++
class Exception {
public:
 Exception();
 Exception(const Exception&);
 virtual ~Exception();
 const Exception& operator=(const Exception&);
};
```

**Notes** CORBA compliant.

**See Also** CORBA::SystemException  
CORBA::UserException  
CORBA::Environment

## CORBA::Exception::Exception()

**Synopsis** CORBA::Exception::Exception();

**Description** Default constructor.

**Notes** CORBA compliant.

**See Also** Other Exception constructors.

## CORBA::Exception::Exception()

**Synopsis** CORBA::Exception::Exception(const CORBA::Exception& e);

**Description** Copy constructor.

**Notes** CORBA compliant.

**See Also** Other Exception constructors.

### **CORBA::Exception::~~Exception()**

- Synopsis** `virtual CORBA::Exception::~~Exception();`
- Description** The destructor is virtual—CORBA::Exception is a base class for system and user-defined exceptions so derived classes may need to provide a destructor to deallocate resources that they have allocated.
- Notes** CORBA compliant.

### **CORBA::Exception::operator=()**

- Synopsis** `const CORBA::Exception& CORBA::Exception::operator = (  
 const CORBA::Exception& e);`
- Description** Assignment operator.
- Notes** Orbix specific.

# CORBA:: ExtraConfigFileCVHandler

## Synopsis

As described in the *Orbix C++ Administrator's Guide*, Orbix provides a configuration file, `iona.cfg`, to configure Orbix. The Orbix configuration handler, `IT_ScopedConfigFile`, reads and writes its values from the default Orbix configuration file. This file is located in the default location for that platform or pointed to by the `IT_CONFIG_PATH` environment variable.

You can provide additional configuration value handlers that read and write to different configuration files by creating an instance of class

`CORBA::ExtraConfigFileCVHandler`. On creation, an instance of this class contains exactly the information stored in the `IT_ScopedConfigFile` handler.

You must activate the new handler using the static function

```
CORBA::ORB::ActivateCVHandler().
```

You can arrange active configuration handlers

explicitly using the static functions `CORBA::ORB::PlaceCVHandlerBefore()` and `CORBA::ORB::PlaceCVHandlerAfter()`. If not explicitly ordered, handlers are called in reverse order of instantiation, that is, the last handler to be instantiated is the first handler to be called.

---

**Note:** If you are migrating from Orbix 2.x and use `PlaceCVHandlerBefore()` or `PlaceCVHandlerAfter()`, you should update your code to specify `IT_ScopedConfigFile` instead of the old `IT_ConfigFile` or `IT_Registry` handlers. Refer to the *Orbix Administrator's Guide* for more details.

---

**Orbix**            `// C++  
class ExtraConfigFileCVHandler {  
public:  
    ExtraConfigFileCVHandler(const char* identifier,  
        const char* fileName);  
    ~ExtraConfigFileCVHandler();  
};`

**Notes**            Orbix specific.

**See Also**        `CORBA::ExtraRegistryFileCVHandler  
CORBA::UserCVHandler  
CORBA::ORB::ActivateCVHandler()`

### **CORBA::ExtraConfigFileCVHandler:: ExtraConfigFileCVHandler()**

**Synopsis**        `CORBA::ExtraConfigFileCVHandler::ExtraConfigFileCVHandler(  
    const char* identifier,  
    const char* fileName);`

**Description**    Constructor.

#### **Parameters**

`identifier`    The name of this configuration value handler.  
`fileName`     The name of the file associated with this handler.

**Notes**            Orbix specific.

### **CORBA::ExtraConfigFileCVHandler:: ~ExtraConfigFileCVHandler()**

**Synopsis**        `virtual  
    CORBA::ExtraConfigFileCVHandler::~~ExtraConfigFileCVHandler();`

**Description**    Destructor.

**Notes**            Orbix specific.



# CORBA::ExtraRegistryCVHandler

**Synopsis** You can configure Orbix using the System Registry on Windows NT and Windows 95. The Orbix configuration handler, `IT_Registry`, reads and writes its values from the System Registry.

You may provide additional configuration value handlers that read and write to a different registry by creating an instance of class

`CORBA::ExtraRegistryFileCVHandler`. On creation, an instance of this class contains exactly the information stored in the `IT_Registry` handler.

The new handler must be activated using the static function

`CORBA::ORB::ActivateCVHandler()`.

Active configuration handlers may be arranged explicitly using the static functions `CORBA::ORB::PlaceCVHandlerBefore()` and

`CORBA::ORB::PlaceCVHandlerAfter()`. If not explicitly ordered, handlers are called in reverse order of instantiation, that is, the last handler to be instantiated is the first handler to be called.

## Orbix

```
// C++
class ExtraRegistryCVHandler {
public:
 ExtraRegistryCVHandler(const char* identifier,
 const char* data);
 ExtraRegistryCVHandler(const char* identifier,
 HKEY hKey);
 ~ExtraRegistryCVHandler();

 HKEY GetRegKey();
};
```

**Notes** Orbix specific.

**See Also** `CORBA::UserCVHandler`  
`CORBA::ExtraConfigFileCVHandler`  
`CORBA::ORB::ActivateCVHandler()`

### **CORBA::ExtraRegistryCVHandler:: ExtraRegistryCVHandler()**

**Synopsis**      `CORBA::ExtraRegistryFileCVHandler::ExtraRegistryCVHandler(  
                  const char* identifier,  
                  const char* data);`

**Description**      Constructor.

**Parameters**

`identifier`      The name of the configuration value handler.  
`data`            An existing sub-key of `HKEY_LOCAL_MACHINE`.

**Notes**            Orbix specific.

**See Also**         Other constructor.

### **CORBA::ExtraRegistryCVHandler:: ExtraRegistryCVHandler()**

**Synopsis**      `CORBA::ExtraRegistryFileCVHandler::ExtraRegistryCVHandler(  
                  const char* identifier,  
                  HKEY hKey);`

**Description**      Constructor.

**Parameters**

`identifier`      The name of the configuration value handler (previously created  
                  with the Windows API function `RegCreateKey()`).  
`hKey`            The registry key for this handler.

**Notes**            Orbix specific.

**See Also**         Other constructor.

### **CORBA::ExtraRegistryCVHandler:: ~ExtraRegistryCVHandler()**

**Synopsis** CORBA::ExtraRegistryFileCVHandler::~~ExtraRegistryCVHandler();  
**Description** Destructor.  
**Notes** Orbix specific.

### **CORBA::ExtraRegistryCVHandler::GetRegKey()**

**Synopsis** HKEY CORBA::ExtraRegistryFileCVHandler::GetRegKey();  
**Description** Returns the registry key for this handler.  
**Notes** Orbix specific.



# CORBA::Filter

**Synopsis** Class CORBA::Filter is a (conceptually) abstract class that describes the interface to a per-process filter.

If you wish to implement a per-process filter you may define a derived class of CORBA::Filter and redefine some or all of the ten monitoring functions as described in the *Orbix Programming Guide*.

**Orbix**

```
// C++
class Filter {
protected:
 Filter();
 virtual ~Filter();
public:
 virtual Boolean outRequestPreMarshal(
 Request&, Environment&);

 virtual Boolean outRequestPostMarshal(
 Request&, Environment&);

 virtual int inRequestPreMarshal(
 Request&, Environment&);

 virtual Boolean inRequestPostMarshal(
 Request&, Environment&);

 virtual Boolean outReplyPreMarshal(
 Request&, Environment&);

 virtual Boolean outReplyPostMarshal(
 Request&, Environment&);

 virtual Boolean inReplyPreMarshal(
 Request&, Environment&);

 virtual Boolean inReplyPostMarshal(
 Request&, Environment&);
```

```
virtual void outReplyFailure(
 Request&, Environment&);

virtual void inReplyFailure(
 Request&, Environment&);
};
```

**Notes** Orbix specific.

**See Also** CORBA::ThreadFilter  
CORBA::AuthenticationFilter

### **CORBA::Filter::Filter()**

**Synopsis** CORBA::Filter::Filter();

**Description** The default constructor adds the newly-created filter object into the per-process filter chain. Direct instances of `Filter` cannot be created: the constructor is protected to enforce this. The derived classes of `Filter` usually have public constructors.

**Notes** Orbix specific.

### **CORBA::Filter::~Filter()**

**Synopsis** virtual CORBA::Filter::~Filter();

**Description** Destructor. Derived classes may need to redefine the destructor.

**Notes** Orbix specific.

### **CORBA::Filter::inReplyFailure()**

**Synopsis** virtual void CORBA::Filter::inReplyFailure(  
 CORBA::Request& r, CORBA::Environment&);

**Description** Defines the action to carry out if the target object raises an exception or if any preceding marshalling filter point ('out request', 'in request', 'out reply' or 'in reply') raises an exception or uses its return value to indicate that the call should not be processed any further.

If not redefined in a derived class, the following implementation is inherited:

```
// C++
{ return; }
```

**Notes** Orbix specific.

### **CORBA::Filter::inReplyPostMarshal()**

**Synopsis** `virtual CORBA::Boolean CORBA::Filter::inReplyPostMarshal(  
CORBA::Request& r, CORBA::Environment&);`

**Description** Defines the action to carry out after any operation on any object in another address space; in particular, after the operation response has arrived at the caller's address space and after the operation's return parameters and return value have been removed from the reply packet.

If not redefined in a derived class, the following implementation is inherited:

```
// C++
{ return 1; } // Continue the call.
```

#### **Return Value**

- 1 Continue with the request as normal. The reply is sent to the next filter on the chain, or if this is the last filter then it is sent to the calling object.
- 0 Do not continue with the call. Reply immediately to the calling object; do not run the remaining filters.

**Exceptions** When redefining this function in a derived class you may raise an exception (and return 0) to indicate that the call is not to be continued. If the return value is 0, and no exception is raised, Orbix raises a `CORBA::FILTER_SUPPRESS` exception with a minor code of `CORBA::FILTER_SUPPRESS_FORCE`.

**Notes** Orbix specific.

### **CORBA::Filter::inReplyPreMarshal()**

**Synopsis**

```
virtual CORBA::Boolean CORBA::Filter::inReplyPreMarshal(
 CORBA::Request& r, CORBA::Environment&);
```

**Description**

Defines the action to perform after any operation on any object in another address space; in particular after the operation response has arrived at the caller's address space and before the operation's return parameters and return value have been removed from the reply packet.

If not redefined in a derived class, the following implementation is inherited:

```
// C++
{ return 1; } // Continue the call.
```

**Return Value**

- 1 Continue with the request as normal. The reply is sent to the next filter on the chain, or if this is the last filter it is sent to the calling object.
- 0 Do not continue with the call. Reply immediately to the calling object; do not run the remaining filters.

**Exceptions**

When redefining this function in a derived class you may raise an exception (and return 0) to indicate that the call is not to be continued. If the return value is 0, and no exception is raised, Orbix raises a `CORBA::FILTER_SUPPRESS` exception with a minor code of `CORBA::FILTER_SUPPRESS_FORCE`.

**Notes**

Orbix specific.

### **CORBA::Filter::inRequestPostMarshal()**

**Synopsis**

```
virtual CORBA::Boolean CORBA::Filter::inRequestPostMarshal(
 CORBA::Request& r, CORBA::Environment&);
```

**Description**

Defines the action to carry out before any incoming operation on any object in the address space; in particular, before the operation has been sent to the target object and after the operation's parameters have been removed from the request packet.

If not redefined in a derived class, the following implementation is inherited:

```
// C++
{ return 1; } // Continue the call.
```



**Return Value**

- 1 Continue with the request as normal. The operation call is sent to the next filter on the chain, or, if this is the last filter, it is sent to the per-object filters, if any, and then to the target object.
- 0 Do not continue with the call. Reply immediately to the caller's address space; do not send the invocation to the target object; do not run the remaining filters.

**Exceptions**

When redefining this function in a derived class you may raise an exception (and return 0) to indicate that the call is not to be continued. If the return value is 0, and no exception is raised, Orbix raises a `CORBA::FILTER_SUPPRESS` exception with a minor code of `CORBA::FILTER_SUPPRESS_FORCE`. The exception is not propagated by Orbix to the caller: at this stage, the invocation is already completed and it is too late to raise an exception.

**Notes**

Orbix specific.

**CORBA::Filter::inRequestPreMarshal()****Synopsis**

```
virtual int CORBA::Filter::inRequestPreMarshal(
 CORBA::Request& r, CORBA::Environment&);
```

**Description**

Defines the action to perform before incoming requests: before any incoming operation on any object in the address space; in particular, before the operation has been sent to the target object and before the operation's parameters have been removed from the request packet.

If not redefined in a derived class, the following implementation is inherited:

```
// C++
{ return 1; } // Continue the call.
```

### Return Value

- 1 Continue with the request as normal. The operation call is sent to the next filter on the chain, or if this is the last filter then it is sent to (the per-object filters and then to) the target object.
- 0 Do not continue with the call. Reply immediately to the caller's address space (where it will be handled by 'in reply' filters); do not run the remaining filters. To indicate that this has occurred, it is recommended that the filter raise an exception for the request.
- 1 The filter has created a thread to handle the request, and this thread may send the request to the target object. The other filters, if any, in the chain are not called. A filter that creates a thread should be placed last in the list of filters. To ensure this, such a filter should inherit from the C++ class `CORBA::ThreadFilter`.

**Exceptions** When redefining this function in a derived class you may raise an exception (and return 0) to indicate that the call is not to be continued. If the return value is 0, and no exception is raised, Orbix raises a `CORBA::FILTER_SUPPRESS` exception with a minor code of `CORBA::FILTER_SUPPRESS_FORCE`.

**Notes** Orbix specific.

**See Also** `CORBA::ThreadFilter`

### **CORBA::Filter::outReplyFailure()**

**Synopsis**

```
virtual void CORBA::Filter::outReplyFailure(
 CORBA::Request& r, CORBA::Environment&);
```

**Description** Defines the action to perform if the target object raises an exception or if any preceding filter point ('in request' or 'out reply') raises an exception or uses its return value to indicate that the call should not be processed any further.

If not redefined in a derived class, the following implementation is inherited:

```
// C++
{ return; }
```

**Notes** Orbix specific.

## CORBA::Filter::outReplyPostMarshal()

**Synopsis**      `virtual CORBA::Boolean CORBA::Filter::outReplyPostMarshal(  
                  CORBA::Request& r, CORBA::Environment&);`

**Description**      Defines the action to perform before outgoing replies: after any incoming operation on any object in the address space; in particular after the operation call has been processed, and after the operation's return parameters and return value have been added to the reply packet.

If not redefined in a derived class, the following implementation is inherited:

```
// C++
{ return 1; } // Continue the call.
```

### Return Value

- 1 Continue with the request as normal. The reply is sent to the next filter on the chain, or if this is the last filter then it is sent to the calling object's address space (where it is handled by 'in reply' filters).
- 0 Do not continue with the call. Reply immediately to the calling object's address space; do not run the remaining filters.

**Exceptions**      When redefining this function in a derived class you may raise an exception (and return 0) to indicate that the call is not to be continued. If the return value is 0, and no exception is raised, Orbix raises a `CORBA::FILTER_SUPPRESS` exception with a minor code of `CORBA::FILTER_SUPPRESS_FORCE`.

**Notes**            Orbix specific.

## CORBA::Filter::outReplyPreMarshal()

**Synopsis**      `virtual CORBA::Boolean CORBA::Filter::outReplyPreMarshal(  
                  CORBA::Request& r, CORBA::Environment&);`

**Description**      Defines the action to perform before outgoing replies: after any incoming operation on any object in the address space; in particular, after the operation call has been processed and before the operation's return parameters and return value have been added to the reply packet.

If not redefined in a derived class, the following implementation is inherited:

```
// C++
{ return 1; } // Continue the call.
```

### Return Value

- 1 Continue with the request as normal. The reply is sent to the next filter on the chain, or, if this is the last filter, it is sent to the calling object's address space (where it will be handled by 'in reply' filters).
- 0 Do not continue with the call. Reply immediately to the calling object's address space; do not run the remaining filters.

### Exceptions

When redefining this function in a derived class you may raise an exception (and return 0) to indicate that the call is not to be continued. If the return value is 0, and no exception is raised, Orbix raises a `CORBA::FILTER_SUPPRESS` exception with a minor code of `CORBA::FILTER_SUPPRESS_FORCE`.

### Notes

Orbix specific.

## **CORBA::Filter::outRequestPostMarshal()**

### Synopsis

```
virtual CORBA::Boolean CORBA::Filter::outRequestPostMarshal(
 CORBA::Request& r, CORBA::Environment&);
```

### Description

Defines the action to perform before outgoing requests: before any operation from this address space to any object in another address space; in particular, before the invocation has been transmitted and after the operation's parameters have been added to the request packet.

If not redefined in a derived class, the following implementation is inherited:

```
// C++
{ return 1; } // Continue the call.
```

### Return Value

- 1 Continue with the request as normal. The operation call is sent to the next filter on the chain, or if this is the last filter it is transmitted to the address space of the target object (where it is first handled by any per-process filters and then per-object filters).

- 0 Do not continue with the call. Reply immediately to the caller; do not send the invocation out of the caller's address space; do not run the remaining filters.

**Exceptions** When redefining this function in a derived class you may raise an exception (and return 0) to indicate that the call is not to be continued. If the return value is 0, and no exception is raised, Orbix raises a `CORBA::FILTER_SUPPRESS` exception with a minor code of `CORBA::FILTER_SUPPRESS_FORCE`.

**Notes** Orbix specific.

### **CORBA::Filter::outRequestPreMarshal()**

**Synopsis**

```
virtual CORBA::Boolean CORBA::Filter::outRequestPreMarshal(
 CORBA::Request& r, CORBA::Environment&);
```

**Description** Defines the action to perform before outgoing requests: before any operation from this address space to any object in another address space; in particular, before the invocation has been transmitted and before the operation's parameters have been added to the request packet.

If not redefined in a derived class, the following implementation is inherited:

```
// C++
{ return 1; } // Continue the call.
```

**Notes** Orbix specific.

#### **Return Value**

- 1 Continue with the request as normal. The operation call is sent to the next filter on the chain, or if this is the last filter it is transmitted to the address space of the target object (where it is first handled by any per-process filters and then per-object filters).
- 0 Do not continue with the call. Reply immediately to the caller; do not send the invocation out of the caller's address space; do not run the remaining filters.

**Exceptions** When redefining this function in a derived class you may raise an exception (and return 0) to indicate that the call is not to be continued. If the return value is 0, and no exception is raised, Orbix raises a `CORBA::FILTER_SUPPRESS` exception with a minor code of `CORBA::FILTER_SUPPRESS_FORCE`.

**Notes** Orbix specific.



# CORBA::Flags

**Synopsis** The member functions of a number of classes—including `CORBA::Context`, `CORBA::Request`, and `CORBA::Object`—take a CORBA specified parameter of type `CORBA::Flags`. In Orbix, `Flags` are manipulated using class `CORBA::Flags`.

**Orbix**

```
// C++
class Flags {
public:
 Flags();
 Flags(ULong i);
 Flags(const Flags& f);

 operator ULong() const { return m_flags;}

 void setf(ULong i);
 void clrf(ULong i);
 void reset();

 int isNil() const;
 int isSet(ULong i) const;
 int isSetAny(ULong i) const;
 int isSetAll(ULong i) const;

 void setArgDef(ULong i);

 Flags& operator=(const Flags& f);
};
```

## CORBA::Flags::Flags()

**Synopsis** `CORBA::Flags::Flags();`

**Description** Default constructor for null flags. All flags are cleared.

**Notes** Orbix specific.

**See Also** Other `Flags` constructors.

### **CORBA::Flags::Flags()**

- Synopsis** `CORBA::Flags::Flags(CORBA::ULong l);`
- Description** Creates a `Flags` object with flags set as indicated by `l`.
- Notes** Orbix specific.
- See Also** Other `Flags` constructors.

### **CORBA::Flags::Flags()**

- Synopsis** `CORBA::Flags::Flags(const CORBA::Flags& f);`
- Description** Copy constructor.
- Notes** Orbix specific.
- See Also** Other `Flags` constructors.

### **CORBA::Flags::operator=()**

- Synopsis** `Flags& operator=(const Flags& f);`
- Description** Assignment operator.
- Notes** Orbix specific.

### **CORBA::Flags::clrf()**

- Synopsis** `void CORBA::Flags::clrf(CORBA::ULong i);`
- Description** Clears flag `i`.
- Notes** Orbix specific.



**CORBA::Flags::isNil()**

**Synopsis** `int CORBA::Flags::isNil() const;`  
**Description** Tests whether or not all flags are clear.  
**Return Value** Returns 1 (TRUE) if all flags clear; returns 0 (FALSE) otherwise.  
**Notes** Orbix specific.

**CORBA::Flags::isSet()**

**Synopsis** `int CORBA::Flags::isSet(CORBA::ULong i) const;`  
**Description** Tests whether flag *i* is set in this object.  
**Return Value** Returns 1 (TRUE) if *i* is set; returns 0 (FALSE) otherwise.  
**Notes** Orbix specific.  
**See Also** `CORBA::Flags::isSetAny()`

**CORBA::Flags::isSetAll()**

**Synopsis** `int CORBA::Flags::isSetAll(CORBA::ULong i) const;`  
**Description** Tests whether all flags set in *i* are set in this object.  
**Return Value** Returns 1 (TRUE) if all flags in *i* are set; returns 0 (FALSE) otherwise.  
**Notes** Orbix specific.

**CORBA::Flags::isSetAny()**

**Synopsis** `int CORBA::Flags::isSetAny(CORBA::ULong i) const;`  
**Description** Tests if any flag set in *i* is set in this object. (This is the same as `isSet(int i)`).  
**Return Value** Returns 1 (TRUE) if any flag in *i* is set; returns 0 (FALSE) otherwise.  
**Notes** Orbix specific.  
**See Also** `CORBA::Flags::isSet()`

### **CORBA::Flags::ULong()**

**Synopsis** `operator CORBA::Flags::ULong() const;`  
**Description** Conversion operator to convert `Flags` object to `CORBA::ULong`.  
**Notes** Orbix specific.

### **CORBA::Flags::reset()**

**Synopsis** `void CORBA::Flags::reset();`  
**Description** Clears all flags.  
**Notes** Orbix specific.

### **CORBA::Flags::setArgDef()**

**Synopsis** `void CORBA::Flags::setArgDef(CORBA::ULong i);`  
**Description** Sets flag as specified in `i`, and clears any other flag. This is usually used to set ARG flags (`CORBA::ARG_IN`, `CORBA::ARG_OUT`, `CORBA::ARG_INOUT`) since they are mutually exclusive.  
**Notes** Orbix specific.  
**See Also** `CORBA::Flags::setf()`

### **CORBA::Flags::setf()**

**Synopsis** `void CORBA::Flags::setf(CORBA::ULong i);`  
**Description** Sets flags as specified in `i`.  
**Notes** Orbix specific.  
**See Also** `CORBA::Flags::setArgDef()`

# CORBA::ImplementationDef

**Synopsis** Class ImplementationDef implements the OMG CORBA pseudo object type ImplementationDef that represents information about an implementation (server).

**CORBA** // IDL  
pseudo interface ImplementationDef {};

**Orbix** // C++  
class ImplementationDef {  
public:  
 static ImplementationDef\_ptr IT\_create(  
 const char\* obj,  
 CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());  
  
 static ImplementationDef\_ptr \_duplicate (  
 ImplementationDef\_ptr obj,  
 CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());  
  
 static ImplementationDef\_ptr \_nil (  
 CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());  
};

**Notes** CORBA compliant.

## CORBA::ImplementationDef::\_duplicate()

**Synopsis** static ImplementationDef\_ptr CORBA::ImplementationDef::\_duplicate(  
 ImplementationDef\_ptr obj,  
 CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());

**Description** Increments the reference count of obj.

**Notes** CORBA compliant.

**See Also** CORBA::release()

### **CORBA::ImplementationDef::\_nil()**

|                    |                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <pre>static ImplementationDef_ptr CORBA::ImplementationDef::_nil(     CORBA::Environment&amp; IT_env = CORBA::IT_chooseDefaultEnv());</pre> |
| <b>Description</b> | Returns a nil object reference for <code>ImplementationDef</code> .                                                                         |
| <b>Notes</b>       | CORBA compliant.                                                                                                                            |
| <b>See Also</b>    | <code>CORBA::is_nil()</code>                                                                                                                |

### **CORBA::ImplementationDef::IT\_create()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <pre>static ImplementationDef_ptr CORBA::ImplementationDef::IT_create(     const char* impl,     CORBA::Environment&amp; IT_env = CORBA::IT_chooseDefaultEnv());</pre>                                                                                                                                                                                                                                         |
| <b>Description</b> | <p>In the absence of a CORBA-specified way to create an <code>ImplementationDef</code> pseudo object in the current standard C++ mapping, Orbix provides the <code>IT_create()</code> function to initialise an object reference for an <code>ImplementationDef</code>.</p> <p>Use of this function is recommended in preference to C++ operator <code>new</code> to ensure memory management consistency.</p> |
| <b>Notes</b>       | CORBA compliant.                                                                                                                                                                                                                                                                                                                                                                                               |

# CORBA::IT\_IOCallback

**Synopsis** Orbix allows a client or server program to receive notification when another a TCP/IP connection to another application is opened or closed. To receive notification of these events, you should define a class that inherits class `CORBA::IT_IOCallback` and implements the functions `OrbixFDOpen()` and `OrbixFDClose()`. Orbix calls these functions when a connection is opened and closed, respectively.

Class `CORBA::IT_IOCallback` also allows you to integrate the Orbix event processing loop with foreign file descriptors. The functions `CORBA::ORB::addForeignFD()` and `CORBA::ORB::addForeignFDSet()` allow you to add foreign file descriptors to the Orbix event loop. If you inherit class `CORBA::IT_IOCallback` and implement the functions `ForeignFDRead()`, `ForeignFDWrite()`, and `ForeignFDExcept()`, you can receive notification when events occur on those foreign file descriptors.

## Orbix

```
// C++
static const unsigned char FD_ORBIX_OPEN;
static const unsigned char FD_ORBIX_CLOSE;

static const unsigned char FD_LOW_WATERMARK;
static const unsigned char FD_STOP_LISTENING;
static const unsigned char FD_START_LISTENING;

static const unsigned char OrbixIO;
static const unsigned char ForeignIO;

static const unsigned char FD_FOREIGN_READ;
static const unsigned char FD_FOREIGN_WRITE;
static const unsigned char FD_FOREIGN_EXCEPT;

class IT_IOCallback {

public:

 virtual ~IT_IOCallback() {}

 virtual void OrbixFDOpen(int fd) {}
```

```
virtual void OrbixFDClose(int fd) {}

virtual void AtOrbixFDLowLimit (int numFDUsed) {}
virtual void StopListeningAtFDHigh (int numFDUsed) {}
virtual void ResumeListeningBelowFDHigh (int numFDUsed) {}

virtual void ForeignFDRead(int fd) {}
virtual void ForeignFDWrite(int fd) {}
virtual void ForeignFDExcept(int fd) {}

};
```

**Notes** Orbix specific.

**See Also**

- CORBA::ORB::addForeignFD()
- CORBA::ORB::addForeignFDSet()
- CORBA::ORB::registerIOCallbackObject()
- CORBA::ORB::removeForeignFD()
- CORBA::ORB::removeForeignFDSet()
- CORBA::ORB::unregisterIOCallbackObject()

### **CORBA::IT\_IOCallback::AtOrbixFDLowLimit()**

**Synopsis**

```
virtual void CORBA::IT_IOCallback::AtOrbixFDLowLimit (
 int numFDUsed);
```

**Description** Defines the action to be taken when the number of file descriptors in use goes above the warning value specified by the configuration variable `Orbix.IT_FD_WARNING_NUMBER`.

#### **Parameters**

`numFDUse` The current number of FD's currently in use by the process  
`d` when the FD warning limit has been exceeded.

**Notes** Orbix specific.

**See Also**

- CORBA::IT\_IOCallbck::StopListeningAtFDHigh()
- CORBA::IT\_IOCallbck::ResumeListeningBelowFDHigh()
- CORBA::ORB::registerIOCallbackObject()
- CORBA::ORB::unregisterIOCallbackObject()

### **CORBA::IT\_IOCallback::ForeignFDExcept()**

|                    |                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <pre>virtual void CORBA::IT_IOCallback::ForeignFDExcept(int fd);</pre>                                                           |
| <b>Description</b> | Defines the action to be taken when an exception event occurs on a foreign file descriptor registered with the Orbix event loop. |
| <b>Parameters</b>  | <p><code>fd</code> The foreign file descriptor on which an event occurred.</p>                                                   |
| <b>Notes</b>       | Orbix specific.                                                                                                                  |

### **CORBA::IT\_IOCallback::ForeignFDRead()**

|                    |                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <pre>virtual void CORBA::IT_IOCallback::ForeignFDRead(int fd);</pre>                                                       |
| <b>Description</b> | Defines the action to be taken when a read event occurs on a foreign file descriptor registered with the Orbix event loop. |
| <b>Parameters</b>  | <p><code>fd</code> The foreign file descriptor on which an event occurred.</p>                                             |
| <b>Notes</b>       | Orbix specific.                                                                                                            |

### **CORBA::IT\_IOCallback::ForeignFDWrite()**

|                    |                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <pre>virtual void CORBA::IT_IOCallback::ForeignFDWrite(int fd);</pre>                                                       |
| <b>Description</b> | Defines the action to be taken when a write event occurs on a foreign file descriptor registered with the Orbix event loop. |
| <b>Parameters</b>  | <p><code>fd</code> The foreign file descriptor on which an event occurred.</p>                                              |
| <b>Notes</b>       | Orbix specific.                                                                                                             |

### **CORBA::IT\_IOCallback::OrbixFDClose()**

**Synopsis** `virtual void CORBA::IT_IOCallback::OrbixFDClose(int fd);`  
**Description** Defines the action to be taken when an existing Orbix connection closes.  
**Parameters**

`fd` The file descriptor associated with the closed connection.

**Notes** Orbix specific.

### **CORBA::IT\_IOCallback::OrbixFDOpen()**

**Synopsis** `virtual void CORBA::IT_IOCallback::OrbixFDOpen(int fd);`  
**Description** Defines the action to be taken when a new Orbix connection is opened.  
**Parameters**

`fd` The file descriptor associated with the new open connection.

**Notes** Orbix specific.

### **CORBA::IT\_IOCallback::ResumeListeningBelowFDHigh()**

**Synopsis** `virtual void CORBA::IT_IOCallback::ResumeListeningBelowFDHigh(int numFDUsed);`  
**Description** Defines the action to be taken when the number of file descriptors in use returns below the limit value specified by the configuration variable `Orbix.IT_FD_STOP_LISTENING_POINT`. The process has now resumed listening.  
**Parameters**

`numFDUsed` The number of FD's currently in use by the process when it has resumed listening.

**Notes** Orbix specific.



**See Also**

```
CORBA::IT_IOCallback::AtOrbixFDLowLimit()
CORBA::IT_IOCallback::StopListeningAtFDHigh()
CORBA::ORB::registerIOCallbackObject()
CORBA::ORB::unregisterIOCallbackObject()
```

### CORBA::IT\_IOCallback::StopListeningAtFDHigh()

**Synopsis**

```
virtual void CORBA::IT_IOCallback::StopListeningAtFDHigh(
 int numFDUsed);
```

**Description**

Defines the action to be taken when the number of file descriptors in use goes above the limit value specified by the configuration variable `Orbix.IT_FD_STOP_LISTENING_POINT`. The process has now stopped listening.

**Parameters**

`numFDUsed` The number of FD's currently in use by the process when it has stopped listening.

**Notes**

Orbix specific.

**See Also**

```
CORBA::IT_IOCallback::AtOrbixFDLowLimit()
CORBA::IT_IOCallback::ResumeListeningBelowFDHigh()
CORBA::ORB::registerIOCallbackObject()
CORBA::ORB::unregisterIOCallbackObject()
```



# CORBA::IT\_reqTransformer

## Synopsis

Class CORBA::IT\_reqTransformer defines the interface for transformer objects that allow a CORBA::Request's data buffer to be modified before an operation invocation is transmitted to a server and before a reply is returned to a client.

If you wish to implement a transformer you may define a derived class of CORBA::IT\_reqTransformer and redefine at least the transform() function as described in the *Orbix Programming Guide*.

## Orbix

```
// C++
class IT_reqTransformer {
protected:
 const char* m_remote_host;
public:
 virtual Boolean transform(
 unsigned char*& data,
 ULong& actual_sz,
 ULong& allocd_sz,
 Boolean send,
 Boolean is_first);

 virtual void free_buf(unsigned char* data,
 ULong actual_sz,
 ULong allocd_sz);

 virtual const char* transform_error();

 void setRemoteHost(const char* host_name);
};

class CORBA::ORB {
public:
 ...
 IT_reqTransformer* setMyReqTransformer(
 IT_reqTransformer*,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
};
```

```
void setReqTransformer(IT_reqTransformer*,
 const char* server,
 const char* host = 0,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

IT_reqTransformer* getMyReqTransformer();
};
```

**Notes** Orbix specific.

**See Also** CORBA::Request

### CORBA::IT\_reqTransformer::free\_buf()

**Synopsis**

```
virtual void CORBA::IT_reqTransformer::free_buf(
 unsigned char* data,
 CORBA::ULong actual_sz,
 CORBA::ULong allocd_sz);
```

**Description** A derived class of `IT_reqTransformer()` that alters the way in which data is stored in an implementation of the `transform()` function may need to provide an implementation for `free_buf()`.

The default implementation of `free_buf()` performs `delete []` on the buffer passed in the parameter `data`.

The function `free_buf()` is called automatically by Orbix after the buffer has been transmitted.

#### Parameters

|                        |                                                                                                                                                                                                                                      |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>data</code>      | The buffer to be freed.                                                                                                                                                                                                              |
| <code>actual_sz</code> | The actual size of the data contained in the buffer. This is not necessarily the same as the size of the memory buffer allocated to store the data since memory buffers are allocated in pages.                                      |
| <code>allocd_sz</code> | Identifies the allocated size of the buffer being passed. This may differ from the actual size of the data buffer. Buffer space is allocated in pages and may, therefore, be larger than the amount of data contained in the buffer. |

**Notes** Orbix specific.

**CORBA::IT\_reqTransformer::transform()****Synopsis**

```
virtual CORBA::Boolean CORBA::IT_reqTransformer::transform(
 unsigned char*& data,
 CORBA::ULong& actual_sz,
 CORBA::ULong& allocd_sz,
 CORBA::Boolean send,
 CORBA::Boolean is_first);
```

**Description**

Defines the transformation to be performed on a `CORBA::Request`. This function is automatically invoked on the registered transformer object immediately prior to transmitting data in a `CORBA::Request` (and after any filtering) and directly subsequent (before any filtering) to receiving data in a `CORBA::Request`.

A derived class of `CORBA::IT_reqTransformer` should override this function to implement a transformation.

An implementation of this function may raise a `TRANSFORM_ERR` system exception to indicate that an error has occurred during the transformation.

**Parameters**

|                        |                                                                                                                                                                                                                       |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>data</code>      | A pointer to the start of the data buffer.                                                                                                                                                                            |
| <code>actual_sz</code> | The size, in bytes, of the actual data contained in the data buffer. This is not necessarily the same as the size of the memory buffer allocated to store the data since memory buffers are allocated in pages.       |
| <code>allocd_sz</code> | The size of the buffer allocated for the <code>Request</code> 's data. Buffer space is allocated in pages and may, therefore, be larger than the amount of data contained in the buffer.                              |
| <code>send</code>      | A flag that identifies whether the data is being transmitted from an address space or received into an address space. A value of 1 indicates that data is outgoing; a value of 0 indicates that the data is incoming. |

`is_first` A flag that identifies whether the data buffer is the first in a list of buffers to be transmitted. This flag is meaningful only if data is being transmitted from an address space since Orbix may allocate more than one buffer to hold data on the sending side. Data received into an address space is placed in a single buffer.

The value of `is_first` is 1 if the parameter `data` points to the first in a list of buffers to be transmitted; otherwise, `is_first` has the value 0.

**Return Value** Returns 1 (`TRUE`) if the transformation is successful, returns 0 (`FASLE`) otherwise. The `TRANSFORM_ERR` system exception is raised by Orbix if 0 is returned.

**Notes** Orbix specific.

**See Also** `CORBA::Filter`

### **CORBA::IT\_reqTransformer::transform\_error()**

**Synopsis**

```
virtual const char* CORBA::IT_reqTransformer::transform_error();
```

**Description** Returns a string describing an error that has occurred in the `transform()` function. A derived class may override this function to return a text string specific to the transformation implemented by the class.

**Notes** Orbix specific. The programmer is responsible for freeing the returned string (using `CORBA::string_free()`).

### CORBA::IT\_reqTransformer::setRemoteHost()

**Synopsis**

```
void CORBA::IT_reqTransformer::setRemoteHost(
 const char* host_name);
```

**Description**

Called by Orbix prior to calling the `transform()` function to record the name of the host initiating the Request.

**Parameters**

`host_name`      The name of the host.

**Notes**

Orbix specific.





# CORBA::LoaderClass

## Synopsis

When an operation invocation arrives at a process, Orbix searches for the target object in the process's object table. By default, if the object is not found, Orbix returns a `CORBA::INV_OBJREF` exception to the caller. However, by installing one or more loader objects in a process, you can choose to intervene and be informed about the failure to find the object.

A loader object might handle such an “object fault” by reconstructing the required object from a persistent store—such as a flat file, an RDBMS, or an ODBMS.

You can then request Orbix to retry the invocation transparently to the caller.

To define a loader, you define a derived class of `CORBA::LoaderClass`. To install a loader, you create a dynamic instance of the new class.

## Orbix

```
// C++
enum saveReason {
 processTermination,
 explicitCall,
 objectDeletion
};

class LoaderClass {
protected:
 LoaderClass(Boolean registerMe = 0);
public:
 virtual ~LoaderClass();

 virtual Object_ptr load(const char* interface,
 const char* marker,
 Boolean isLocal, Environment&);

 virtual void save(Object_ptr obj, saveReason reason,
 Environment&);

 virtual void record(Object_ptr obj, char*& marker,
 Environment&);
};
```

```
 virtual Boolean rename(Object_ptr obj, char*& marker,
 Environment&);
};
```

**Notes** Orbix specific.

**See Also** CORBA::NullLoaderClass

### **CORBA::LoaderClass::LoaderClass()**

**Synopsis** CORBA::LoaderClass::LoaderClass(CORBA::Boolean registerMe = 0);

**Description** The `LoaderClass` constructor has protected access—you cannot, therefore, create a direct instance of class `LoaderClass`.

#### **Parameters**

`registerMe` The value of this parameter must be 1 (`TRUE`) if the `load()` function of the new loader is to be called by Orbix, rather than explicitly by you.

**Notes** Orbix specific.

**See Also** CORBA::LoaderClass::load()

### **CORBA::LoaderClass::~~LoaderClass()**

**Synopsis** virtual CORBA::LoaderClass::~~LoaderClass();

**Description** The destructor.

**Notes** Orbix specific.

---

## CORBA::LoaderClass::load()

**Synopsis**

```
virtual CORBA::Object_ptr CORBA::LoaderClass::load(
 const char* interface,
 const char* marker, CORBA::Boolean isLocal);
```

**Description**

When an object fault occurs, the `load()` function is called on each loader in turn until one of them successfully returns the address of the object, or until they have all returned zero.

The responsibility of the `load()` function is to determine if the required object is to be loaded by the current loader, and if so, then to create the object and assign the correct marker to it.

**Parameters**

`interface` The interface name of the missing object is determined as follows: if an object fault occurs during the call:

```
// C++
pPtr = I1::_bind(<parameters>);
the interface name in load() is "I1".
```

If the first parameter to `_bind()` is a full object reference string, Orbix returns an exception if the reference's interface field is not I1 or a derived interface of I1.

If an object fault occurs during the call:

```
// C++
pPtr = CORBA::Orbix.string_to_object
(<full object reference string>);
```

the interface name in `load()` is that extracted from the full object reference string.

If a loader is called because of a reference entering an address space (as an `in`, `out` or `inout` parameter, a return value, or as the target object of an operation call), the interface name in `load()` is the interface name extracted from the object reference.

The switches passed to the IDL compiler affect how the interface name is seen by `load()`. Refer to `CORBA::Object::_interfaceMarker()`.

`marker`      The marker of the required object.

`isLocal`     Set to 1 (TRUE) if the object fault occurred because of a call to `_bind()` or `CORBA::Orbix.string_to_object()` by the process itself.

                 Set to 0 (FALSE) if the object fault occurred because of an object fault on the target object of an incoming operation invocation, or on an `in`, `out` or `inout` parameter or return value.

**Return Value**   Returns the object reference if the object is found, returns a nil object reference otherwise.

**Notes**            Orbix specific.

**See Also**        `CORBA::ORB::string_to_object()`  
`CORBA::LoaderClass::save()`  
`CORBA::Object::_interfaceMarker()`

### **CORBA::LoaderClass::record()**

**Synopsis**

```
virtual void CORBA::LoaderClass::record(
 CORBA::Object_ptr obj_ref
 char*& marker);
```

**Description**    When you name an object by passing a marker name to the TIE or BOAImpl constructor, Orbix calls `record()` on the object's loader.

A derived class may redefine `record()` to override your choice of name.

The default loader, implemented by `CORBA::NullLoaderClass`, inherits its implementation of `record()` from `LoaderClass`. This implementation does not change the chosen name. It may, however, raise an exception if the name is in use (that is, an object with the same interface name and marker name already exists in the server process) and the object consequently can not be registered.

If no marker name is suggested (`marker` is zero), the default `NullLoaderClass` `record()` function chooses a name that is a string of decimal digits, different to any generated before in the current process.

You may also name an object using the function `CORBA::Object::_marker()`. In the case, Orbix calls `rename()` rather than `record()` on the object's loader.

**Notes**            Orbix specific.

**See Also** CORBA::LoaderClass::rename()  
CORBA::NullLoaderClass

### CORBA::LoaderClass::rename()

**Synopsis** virtual CORBA::Boolean CORBA::LoaderClass::rename(  
CORBA::Object\_ptr obj,  
char\*& marker);

**Description** When you name an object by calling CORBA::Object::\_marker(), Orbix calls rename() on the object's loader.

A derived class may redefine rename() to override your choice of name.

The default loader, implemented by CORBA::NullLoaderClass, inherits its implementation of rename() from LoaderClass. This implementation does not change the chosen name. It may, however, raise an exception, and return 0, if the name is in use (that is, an object with the same interface name and marker name already exists in the server process).

You may also name an object by passing a marker name to the TIE or BOAImpl constructor. In this case, Orbix calls record() on the object's loader.

**Return Value** Returns 1 (TRUE) if the object is successfully renamed, returns 0 (FALSE) otherwise.

**Notes** Orbix specific.

**See Also** CORBA::LoaderClass::record()  
CORBA::Object::\_marker()

### CORBA::LoaderClass::save()

**Synopsis** virtual void CORBA::LoaderClass::save(  
CORBA::Object\_ptr obj,  
saveReason reason);

**Description** When a process terminates, Orbix iterates through all of the objects in its object table and calls save() on the loader associated with each object. A loader may save the object to persistent storage (either by calling a function on the object, or by accessing the object's data).

The associated loader's `save()` function is also called when an object is destroyed; and you can also call it explicitly for an object by calling its `CORBA::Object::_save()` function. `CORBA::Object::_save()` simply calls the `save()` function on the object's loader. You must call the `_save()` function in the same address space as the target object: calling it in a client process, that is, on a proxy, has no effect.

As Orbix iterates through its object table on process termination, it calls `save()` on each object's loader; note, however, that it does not destroy the objects themselves. It does destroy the loader objects afterwards.

### Parameters

- `obj` The object on whose loader is `save()` is being called.
- `reason` The reason that `save()` has been called. This may be:
- `processTermination`: The process is about to exit.
  - `explicitCall`: The object's `_save()` function has been called.
  - `objectDeletion`: `CORBA::release()` has been called on the object, which previously had a reference count of 1.

**Notes** Orbix specific.

**See Also** `CORBA::NullLoaderClass`  
`CORBA::Object::_save()`  
`CORBA::LoaderClass::load()`

# CORBA::NamedValue

**Synopsis** The C++ class CORBA::NamedValue implements the IDL pseudo object type NamedValue that is used only as an element of an NVList, chiefly in the DII. A NamedValue describes an argument to a Request: it contains an optional name, an any value and labelling flags.

**CORBA**

```
// Pseudo IDL
pseudo interface NamedValue {
 readonly attribute Identifier name;
 readonly attribute any value;
 readonly attribute Flags flags;
};
```

**Orbix**

```
// C++
typedef char* Identifier;

class NamedValue : public IT_PseudoIDL {
public:
 const char* name() const;
 Any* value ()const;
 Flags flags ()const;

 NamedValue();
 NamedValue(const NamedValue&);
 const NamedValue& operator=(const NamedValue&);

 ~NamedValue();

 static NamedValue_ptr IT_create(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
 static NamedValue_ptr IT_create(const NamedValue&,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

 static NamedValue_ptr _duplicate(
 NamedValue_ptr obj,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

```
static NamedValue_ptr _nil(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
};
```

**Notes** CORBA compliant.

**See Also** CORBA::NVList  
CORBA::Request  
CORBA::Flags

### **CORBA::NamedValue::NamedValue()**

**Synopsis** CORBA::NamedValue::NamedValue();

**Description** Default constructor.

**Notes** Orbix specific. See CORBA::NVList::add(), CORBA::NVList::add\_item(), CORBA::NVList::add\_value(), CORBA::NVList::add\_item\_consume(), CORBA::NVList::add\_value\_consume() and CORBA::ORB::create\_named\_value() for CORBA compliant ways to create a NamedValue.

**See Also** CORBA::NVList::add()  
CORBA::NVList::add\_item()  
CORBA::NVList::add\_value()  
CORBA::NVList::add\_item\_consume()  
CORBA::NVList::add\_value\_consume()  
CORBA::NamedValue::IT\_create()  
CORBA::ORB::create\_named\_value()

### **CORBA::NamedValue::NamedValue()**

**Synopsis** CORBA::NamedValue::NamedValue(const CORBA::NamedValue& nv);

**Description** Copy constructor.

**Notes** Orbix specific.

**See Also** CORBA::NVList::add()  
CORBA::NVList::add\_item()  
CORBA::NVList::add\_value()  
CORBA::NVList::add\_item\_consume()  
CORBA::NVList::add\_value\_consume()



---

```
CORBA::NamedValue::IT_create()
CORBA::ORB::create_named_value()
```

### CORBA::NamedValue::~~NamedValue()

**Synopsis** `CORBA::NamedValue::~~NamedValue();`

**Description** Destructor.

**Notes** Orbix specific.

**See Also** `CORBA::release()`

### CORBA::NamedValue::IT\_create()

**Synopsis**

```
static NamedValue_ptr IT_create(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
static NamedValue_ptr IT_create(const NamedValue&,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** For consistency with other pseudo object types, Orbix provides the `IT_create()` function for class `NamedValue` in order to obtain a `NamedValue` object reference. To ensure memory management consistency, you should not use the C++ `new` operator to create a `NamedValue`.

See the corresponding constructor for details of the parameters to `IT_create()`.

**Notes** Orbix specific. See `CORBA::NVList::add()`, `CORBA::NVList::add_item()`, `CORBA::NVList::add_value()`, `CORBA::NVList::add_item_consume()`, `CORBA::NVList::add_value_consume()` and `CORBA::ORB::create_named_value()` for CORBA compliant ways to create a `NamedValue`.

**See Also**

```
CORBA::NVList::add()
CORBA::NVList::add_item()
CORBA::NVList::add_value()
CORBA::NVList::add_item_consume()
CORBA::NVList::add_value_consume()
CORBA::NamedValue::IT_create()
CORBA::ORB::create_named_value()
```

### **CORBA::NamedValue::\_duplicate()**

- Synopsis**        `static CORBA::NamedValue_ptr CORBA::NamedValue::_duplicate(  
                  CORBA::NamedValue_ptr obj,  
                  CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description**    Increments the reference count of obj.
- Return Value**   Returns a reference to self.
- Notes**         CORBA compliant.
- See Also**       CORBA::release()

### **CORBA::NamedValue::flags()**

- Synopsis**        `CORBA::Flags CORBA::NamedValue::flags() const;`
- Description**    Returns the CORBA::Flags object associated with the NamedValue.
- Notes**         CORBA compliant.
- See Also**       CORBA::Flags

### **CORBA::NamedValue::name()**

- Synopsis**        `const char* CORBA::NamedValue::name() const;`
- Description**    The optional name associated with the NamedValue. This is the name of a parameter or argument to a request.
- Return Value**   The return value is a pointer to the internal memory of the NamedValue.
- Notes**         CORBA compliant.

**CORBA::NamedValue::\_nil()**

- Synopsis**      `static NamedValue_ptr CORBA::NamedValue::_nil(  
                  CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description**    Returns a nil object reference for a NamedValue.
- Notes**          CORBA compliant.
- See Also**        `CORBA::is_nil()`

**CORBA::NamedValue::operator=()**

- Synopsis**        `const CORBA::NamedValue& CORBA::NamedValue::operator=(  
                  const CORBA::NamedValue& nv);`
- Description**    Assignment operator.
- Notes**          Orbix specific.

**CORBA::NamedValue::value()**

- Synopsis**        `CORBA::Any* CORBA::NamedValue::value() const;`
- Description**    Returns a pointer to the `CORBA::Any` contained in the `NamedValue`.
- Return Value**   The return value is a pointer to the internal memory of the `NamedValue`.
- Notes**          CORBA compliant.
- See Also**        `CORBA::Any`



# CORBA::NullLoaderClass

**Synopsis** The default loader is an instance of a CORBA::NullLoaderClass. This class inherits the functions load(), save() and rename() from CORBA::LoaderClass. The default loader does not support persistence. It is associated with all objects that are not explicitly associated with another loader.

**Orbix**

```
// C++
// In namespace CORBA.
class NullLoaderClass : public LoaderClass {
public:
 NullLoaderClass();

 virtual void record(Object_ptr obj, char*& marker,
 Environment&);
};
static NullLoaderClass* defaultLoader;
```

**Notes** Orbix specific.

**See Also** CORBA::LoaderClass

## CORBA::NullLoaderClass::NullLoaderClass()

**Synopsis** CORBA::NullLoaderClass::NullLoaderClass();

**Description** Default constructor.

**Notes** Orbix specific.

**See Also** CORBA::LoaderClass::LoaderClass()

### **CORBA::NullLoaderClass::record()**

**Synopsis**

```
virtual void CORBA::NullLoaderClass::record(
 CORBA::Object_ptr obj_ref obj,
 char*& marker, CORBA::Environment&);
```

**Description**

If no marker name is suggested in `marker`, this function chooses one that is a string of decimal digits, different to any it generated before.

You can ensure that the markers they choose are different from those chosen by Orbix by not using strings that consist entirely of digits.

**Notes**

Orbix specific.

**See Also**

`CORBA::LoaderClass::record()`

# CORBA::NVList

## Synopsis

The C++ class CORBA::NVList implements the CORBA pseudo object type NVList. An NVList is a list of NamedValue elements—a NamedValue describes an argument to a Request.

## CORBA

```
// Pseudo IDL
exception Bounds {};
pseudo interface NVList {
 readonly attribute unsigned long count;
 NamedValue add(in Flags flags);
 NamedValue add_item(in Identifier item_name,
 in Flags flags);
 NamedValue add_value(in Identifier item_name,
 in any val, in Flags flags);
 NamedValue unsigned item(in long index) raises(Bounds);
 Status remove(in unsigned long index) raises(Bounds);
};
```

## Orbix

```
// C++
struct NVList : public IT_PseudoIDL {
 ULong count(CORBA::Environment& IT_env =
CORBA::IT_chooseDefaultEnv()) const;

 NamedValue_ptr add(
 Flags item_flags,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

 NamedValue_ptr add_item(
 const char* item_name,
 Flags item_flags
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

 NamedValue_ptr add_item_consume(
 char* item_name,
 Flags item_flags,);

 NamedValue_ptr add_value_consume(
 char* item_name,
 Any* item_value,
```

```
Flags item_flags,);

NamedValue_ptr add_value(
 const char* item_name,
 const Any& item_type,
 Flags item_flags,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

NamedValue_ptr item(
 CORBA::ULong index,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

Status remove(CORBA::Long l,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

NVList();

NVList(Long l, CORBA::Environment& IT_env =
CORBA::IT_chooseDefaultEnv());

NVList(const NVList&);

const NVList& operator=(const NVList&);

~NVList();

static NVList_ptr IT_create(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

static NVList_ptr IT_create(
 Long,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

static NVList_ptr IT_create(
 const NVList&,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

static NVList_ptr _duplicate(
 NVList_ptr obj,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

static NVList_ptr _nil(
```



```

CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
};

```

**Notes** CORBA compliant.

**See Also** CORBA::NamedValue  
CORBA::Request

### CORBA::NVList::NVList()

**Synopsis** CORBA::NVList::NVList();

**Description** Default constructor.

**Notes** Orbix specific. Refer to CORBA::ORB::create\_list() and CORBA::ORB::create\_operation\_list() for CORBA compliant ways to create an NVList.

**See Also** CORBA::ORB::create\_list()  
CORBA::ORB::create\_operation\_list()  
CORBA::NVList::IT\_create()

Other NVList constructors.

### CORBA::NVList::NVList()

**Synopsis** CORBA::NVList::NVList(CORBA::Long size,  
CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());

**Description** Constructs an NVList of length size.

**Notes** Orbix specific. See CORBA::ORB::create\_list() and CORBA::ORB::create\_operation\_list() for CORBA compliant ways to create an NVList.

**See Also** CORBA::ORB::create\_list()  
CORBA::ORB::create\_operation\_list()  
CORBA::NVList::IT\_create()

Other NVList constructors.

### **CORBA::NVList::NVList()**

|                    |                                                               |
|--------------------|---------------------------------------------------------------|
| <b>Synopsis</b>    | <code>CORBA::NVList::NVList(const CORBA::NVList&amp;);</code> |
| <b>Description</b> | Copy constructor.                                             |
| <b>Notes</b>       | Orbix specific.                                               |
| <b>See Also</b>    | Other <code>NVList</code> constructors.                       |

### **CORBA::NVList::~~NVList()**

|                    |                                         |
|--------------------|-----------------------------------------|
| <b>Synopsis</b>    | <code>CORBA::NVList::~~NVList();</code> |
| <b>Description</b> | Destructor.                             |
| <b>Notes</b>       | Orbix specific.                         |

### **CORBA::NVList::operator=()**

|                    |                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <code>const CORBA::NVList&amp; CORBA::NVList::operator = (<br/>const CORBA::NVList&amp;);</code> |
| <b>Description</b> | Assignment operator.                                                                             |
| <b>Notes</b>       | Orbix specific.                                                                                  |

### **CORBA::NVList::\_duplicate()**

|                     |                                                                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>     | <code>static CORBA::NVList_ptr CORBA::NVList::_duplicate(<br/>CORBA::NVList_ptr obj,<br/>CORBA::Environment&amp; IT_env = CORBA::IT_chooseDefaultEnv());</code> |
| <b>Description</b>  | Increments the reference count of <code>obj</code> .                                                                                                            |
| <b>Return Value</b> | Returns a reference to self.                                                                                                                                    |
| <b>Notes</b>        | CORBA compliant.                                                                                                                                                |
| <b>See Also</b>     | <code>CORBA::release()</code>                                                                                                                                   |

---

### CORBA::NVList::\_nil()

- Synopsis**      `static CORBA::NVList_ptr CORBA::NVList::_nil(  
                  CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description**    Returns a nil object reference for an NVList object.
- Notes**          CORBA compliant.
- See Also**        `CORBA::is_nil()`

### CORBA::NVList::add()

- Synopsis**      `CORBA::NamedValue_ptr CORBA::NVList::add(  
                  CORBA::Flags item_flags,  
                  CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description**    Creates an unnamed NamedValue, initialising only the Flags and adds it to the list.
- Parameters**
- `item_flags`    Item flags (ARG\_IN, ARG\_OUT, ARG\_INOUT).
- Return Value**   Returns the new NamedValue. The reference count of the returned NamedValue pseudo object is *not* incremented. Therefore, the caller should not release the returned reference when no longer needed, nor assign it to a `_var` variable.
- Notes**          CORBA compliant.

### CORBA::NVList::add\_item()

- Synopsis**      `CORBA::NamedValue_ptr CORBA::NVList::add_item(  
                  const char* item_name,  
                  CORBA::Flags item_flags,  
                  CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description**    Creates a NamedValue with name and Flags initialised, and adds it to the list.
- Parameters**
- `item_name`    Name of item.

`item_flags` Item flags (ARG\_IN, ARG\_OUT, ARG\_INOUT).

**Return Value** Returns the new `NamedValue`. The reference count of the returned `NamedValue` pseudo object is *not* incremented. Therefore, the caller should not release the returned reference when no longer needed, nor assign it to a `_var` variable.

**Notes** CORBA compliant.

### **CORBA::NVList::add\_value()**

**Synopsis**

```
CORBA::NamedValue_ptr CORBA::NVList::add_value(
 const char* item_name,
 const CORBA::Any& value,
 CORBA::Flags item_flags,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Creates a `NamedValue` with name, value and flags initialised and adds it to the list.

#### **Parameters**

`item_name` Name of item.

`value` Value of item.

`item_flags` Item flags (ARG\_IN, ARG\_OUT, ARG\_INOUT).

**Return Value** Returns the new `NamedValue`. The reference count of the returned `NamedValue` pseudo object is *not* incremented. Therefore, the caller should not release the returned reference when no longer needed, nor assign it to a `_var` variable.

**Notes** CORBA compliant.

### **CORBA::NVList::add\_item\_consume()**

**Synopsis**

```
CORBA::NamedValue_ptr CORBA::NVList::add_item_consume(
 char* item_name,
 CORBA::Flags item_flags);
```

**Description** Creates a `NamedValue` with name and flags initialised, and adds it to the list. The `char*` parameter is consumed by the `NVList`. The caller may not access this data after it has been passed to this function.

**Parameters**

`item_name` Name of item.  
`item_flags` Item flags (ARG\_IN, ARG\_OUT, ARG\_INOUT).

**Return Value** Returns the new `NamedValue`. The reference count of the returned `NamedValue` pseudo object is *not* incremented. Therefore, the caller should not release the returned reference when no longer needed, nor assign it to a `_var` variable.

**Notes** CORBA compliant.

**CORBA::NVList::add\_value\_consume()****Synopsis**

```
CORBA::NamedValue_ptr CORBA::NVList::add_value_consume(
 char* item_name,
 CORBA::Any* item_value,
 CORBA::Flags item_flags);
```

**Description**

Creates a `NamedValue` with name, values and flags initialised, and adds it to the list. The `char*` and the `Any*` parameters are consumed by the `NVList`. The caller may not access this data after it has been passed to this function. The caller should use the `NamedValue::value ()` operation to modify the value attribute of the underlying `NamedValue`, if desired.

**Parameters**

`item_name` Name of item.  
`item_value` Value of item.  
`item_flags` Item flags (ARG\_IN, ARG\_OUT, ARG\_INOUT).

**Return Value** Returns the new `NamedValue`. The reference count of the returned `NamedValue` pseudo object is *not* incremented. Therefore, the caller should not release the returned reference when no longer needed, nor assign it to a `_var` variable.

**Notes** CORBA compliant.

**CORBA::NVList::count()****Synopsis**

```
CORBA::ULong CORBA::NVList::count (
```

```
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;
```

**Description** Returns the number of elements in the `NVList`.

**Notes** CORBA compliant.

### **CORBA::NVList::IT\_create()**

**Synopsis**

```
static CORBA::NVList_ptr CORBA::NVList::IT_create(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
static CORBA::NVList_ptr CORBA::NVList::IT_create(
 CORBA::ULong,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
static CORBA::NVList_ptr CORBA::NVList::IT_create(
 const CORBA::NVList&,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** For consistency with other pseudo object types for which there is no CORBA specified way in the current C++ mapping to obtain an object reference, Orbix provides the `IT_create()` functions for class `NVList`. To ensure memory management consistency, you should not use the C++ `new` operator to create an `NVList`.

Refer to the corresponding constructors for details of the parameters to `IT_create()`.

**Notes** Orbix specific. See `CORBA::ORB::create_list()` and `CORBA::ORB::create_operation_list()` for CORBA compliant ways of creating an `NVList`.

**See Also** `CORBA::ORB::create_list()`  
`CORBA::ORB::create_operation_list()`  
`NVList` constructors.

### **CORBA::NVList::item()**

**Synopsis**

```
CORBA::NamedValue_ptr CORBA::NVList::item(CORBA::ULong index,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Returns the list item at the given index. The first item is at index 0.

**Exceptions**      Raises a `Bounds` exception if `index` is out of range.

**Notes**            CORBA compliant.

### **CORBA::NVList::remove()**

**Synopsis**            `CORBA::Status CORBA::NVList::remove(CORBA::ULong index,`  
                         `CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description**        Removes the item at the given `index`. The first item is at `index 0`.

**Exceptions**        Raises a `Bounds` exception if `index` is out of range.

**Notes**            CORBA compliant.





# CORBA::NVListIterator

**Synopsis** Defines a C++ iterator for CORBA::NVList that returns the NamedValues in the list.

**Orbis**

```
// C++
class NVListIterator {
public:
 NVListIterator();
 NVListIterator(NVList_ptr l);

 void setList(NVList_ptr l);
 NamedValue_ptr operator() ();
};
```

**Notes** Orbix specific.

**See Also** CORBA::NVList  
CORBA::NamedValue

## CORBA::NVListIterator::NVListIterator()

**Synopsis** CORBA::NVListIterator::NVListIterator();

**Description** Constructor. The iterator may be associated with an NVList using setList().

**Notes** Orbix specific.

**See Also** CORBA::NVListIterator::setList()  
Other NVListIterator constructor.

### **CORBA::NVListIterator::NVListIterator()**

- Synopsis** `CORBA::NVListIterator::NVListIterator(CORBA::NVList_ptr l);`
- Description** Constructor. Creates an iterator for `NVList l`.
- Notes** Orbix specific.
- See Also** Other `NVListIterator` constructor.

### **CORBA::NVListIterator::operator()**

- Synopsis** `CORBA::NamedValue_ptr CORBA::NVListIterator::operator()();`
- Description** When called, successively returns each of the `NamedValue` elements in the `NVList` over which it iterates.
- Notes** Orbix specific.

### **CORBA::NVListIterator::setList()**

- Synopsis** `CORBA::NVListIterator::setList(CORBA::NVList_ptr l);`
- Description** Sets the iterator to the start of list `l`. The list `l` is not copied.
- Notes** Orbix specific.

# CORBA::Object

## Synopsis

CORBA::Object is the base class for all IDL C++ classes. Each BOAImpl class and each TIE class inherits from an IDL C++ class and hence also from CORBA::Object. On the client side, the functions of CORBA::Object are called on a proxy (unless collocation is set); on the server side, they are called on the real object.

All objects of type CORBA::Object hold their own full object reference in their member data. CORBA::Object provides functions to retrieve object reference fields in string format and to convert between object references and strings. It also provides functions to manipulate per-object filters, create Request objects for the DII and so on. The `is_nil()` and `release()` operations of interface Object are provided in the CORBA namespace in the mapped C++ class. All other operations have leading underscores in the C++ mapping.

## CORBA

```
// IDL
interface Object {
 boolean is_nil();
 Object duplicate();
 void release();
 ImplementationDef get_implementation();
 InterfaceDef get_interface();
 Status create_request(
 in Context ctx,
 in Identifier operation,
 in NVList arg_list,
 in NamedValue result,
 out Request request,
 in Flags req_flags);
};
```

## Orbix

```
// C++
typedef char* Identifier;

enum ProtocolKind
{
 IT_ORBIX_OR_KIND = 0,
 IT_INTEROPERABLE_OR_KIND = 1,
 ITNEO_OR_KIND = 2
};
```

```
};

typedef ULong ORBStatus;
typedef Object_ptr ObjectRef;

class Object {
public:

 Object(Boolean normal = 0);

 Object(const Object&);

 Object(const char*);

 Object(const char* host,
 const char* impl,
 const char* marker,
 const char* intfHost,
 const char* intfImpl,
 const char* intfMarker);

 Object(const ObjectReferenceImpl*);

 virtual ~Object();

 ImplementationDef_ptr _get_implementation(
 Environment& IT_env =
 IT_chooseDefaultEnv());

 InterfaceDef_ptr _get_interface(
 Environment& IT_env =
 IT_chooseDefaultEnv());

 Request_ptr _request(
 const char*,
 Environment& IT_env =
 IT_chooseDefaultEnv());

 ORBStatus _create_request(
 Context_ptr ctx,
 const char* operation,
 NVList_ptr arg_list,
 NamedValue_ptr& result,
```

```
Request_ptr& request,
Flags req_flags,
Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean _is_a(
 const char* logical_type_id,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean _non_existent(
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean _is_equivalent(
 const ObjectRef,
 Environment& IT_env =
 IT_chooseDefaultEnv());

ULong _hash(
 ULong maximum,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Identifier _object_to_string(
 unsigned long kind = IT_ORBIX_OR_KIND,
 Environment& IT_env =
 IT_chooseDefaultEnv());

const char* _host(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

const char* _implementation(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

const char* _marker(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

const char* _interfaceHost(
 Environment& IT_env =
```

```
IT_chooseDefaultEnv() const;

const char* _interfaceImplementation(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

const char* _interfaceMarker(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

Boolean _isNullProxy(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

Boolean _isNull(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

void* _attachPre(void* b);

void* _attachPost(void* b);

void* _getPre() const;

void* _getPost() const;

virtual void* _deref();

Boolean _enableInternalLock(
 Boolean isEnabled,
 Environment& IT_env =
 IT_chooseDefaultEnv());

int _fd(
 Environment& IT_env =
 IT_chooseDefaultEnv());

virtual LoaderClass* _loader(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

virtual void _save(
 Environment& IT_env =
```

---

```
 IT_chooseDefaultEnv());

virtual Boolean _isRemote(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

ULong _refCount(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

void _closeChannel(
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean _hasValidOpenChannel(
 Environment& IT_env =
 IT_chooseDefaultEnv());

static Object_ptr _duplicate(
 Object_ptr obj,
 Environment& IT_env =
 IT_chooseDefaultEnv());

static Object_ptr _nil(
 Environment& IT_env =
 IT_chooseDefaultEnv());

virtual void _release(
 Environment& IT_env =
 IT_chooseDefaultEnv());

};
```

**Notes**

CORBA compliant.

### **CORBA::Object::Object()**

|                    |                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <code>CORBA::Object::Object(CORBA::Boolean normal = 0 (FALSE));</code>                                |
| <b>Description</b> | Default constructor. If the parameter <code>normal</code> is set to 1 (TRUE) a null proxy is created. |
| <b>Notes</b>       | Orbix specific. You should not normally need to use this function.                                    |
| <b>See Also</b>    | Other <code>Object</code> constructors.                                                               |

### **CORBA::Object::Object()**

|                    |                                                                   |
|--------------------|-------------------------------------------------------------------|
| <b>Synopsis</b>    | <code>CORBA::Object::Object(const CORBA::Object&amp; obj);</code> |
| <b>Description</b> | Copy constructor.                                                 |
| <b>Notes</b>       | Orbix specific.                                                   |
| <b>See Also</b>    | Other <code>Object</code> constructors.                           |

### **CORBA::Object::Object()**

|                    |                                                                      |
|--------------------|----------------------------------------------------------------------|
| <b>Synopsis</b>    | <code>CORBA::Object::Object(const CORBA::Object_ptr obj_ptr);</code> |
| <b>Description</b> | Conversion from an <code>Object_ptr</code> .                         |
| <b>Notes</b>       | Orbix specific.                                                      |
| <b>See Also</b>    | Other <code>Object</code> constructors.                              |

### **CORBA::Object::Object()**

|                 |                                                                                                                                                                                     |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b> | <code>CORBA::Object::Object(const char* host,<br/>const char* impl,<br/>const char* marker,<br/>const char* intfHost,<br/>const char* intfImpl,<br/>const char* intfMarker);</code> |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|



**Description** Constructs an object from the given as arguments. Note that an object created using this constructor cannot subsequently be narrowed using `_narrow()`; you should use the CORBA compliant function `CORBA::ORB::string_to_object()` instead.

**Parameters**

|                         |                                                                                                    |
|-------------------------|----------------------------------------------------------------------------------------------------|
| <code>host</code>       | The host name of the target object.                                                                |
| <code>impl</code>       | The name of the target object's server.                                                            |
| <code>marker</code>     | The object's marker name.                                                                          |
| <code>intfHost</code>   | The name of a host running an Interface Repository that stores the target object's IDL definition. |
| <code>intfImpl</code>   | The string "IFR".                                                                                  |
| <code>intfMarker</code> | The target object's true (most derived) interface.                                                 |

**Notes** Orbix specific. The CORBA compliant version of this function is `CORBA::ORB::string_to_object()`.

**See Also** `CORBA::ORB::string_to_object()`  
`CORBA::ORB::object_to_string()`  
`CORBA::BOA::create()`  
Other Object constructors.

### **CORBA::Object::~~Object()**

**Synopsis** `virtual CORBA::Object::~~Object();`

**Description** Destructor.

**Notes** Orbix specific.

### **CORBA::Object::\_attachPost()**

**Synopsis** `void* CORBA::Object::_attachPost(void* f);`

**Description** Attaches a per-object post-filter to the target object. Attaching a post-filter to an object that already has a post-filter causes the old filter to be removed and the new one attached. To attach a chain of per-object post-filters to an object, you can use

`_attachPost()` to attach the first post-filter, and then you can use it again to attach a post-filter to the first filter and so on. Thus, `_attachPost()` should be called on *filter* objects to create a chain.

### Parameters

- `f` A pointer to a filter object. The dynamic type of `f` should be a class that implements the IDL interface of the object on which `_attachPost()` is invoked.

**Return Value** Returns a pointer to the previous pre-filter, if any, attached to the object.

**Notes** Orbix specific.

**See Also** `CORBA::Object::_getPost()`  
`CORBA::Object::_attachPre()`  
`CORBA::Filter`

## CORBA::Object::\_attachPre()

**Synopsis** `void* CORBA::Object::_attachPre(void* f);`

**Description** Attaches a per-object pre-filter to the target object. Attaching a pre-filter to an object that already has a pre-filter causes the old filter to be removed and the new one attached. To attach a chain of per-object pre-filters to an object, you can use `_attachPre()` to attach the first pre-filter, and then you can use it again to attach a pre-filter to the first filter and so on. Thus, `_attachPre()` should be called on *filter* objects to create a chain.

### Parameters

- `f` A pointer to a filter object. The dynamic type of `f` should be a pointer to an object that implements the IDL interface of the object on which `_attachPre()` is invoked.

**Return Value** Returns a pointer to the previous pre-filter, if any, attached to the object.

**Exceptions** If a per-object pre-filter raises an exception, the operation invocation call is not passed to the target object. Normally this exception is returned to the client to indicate the outcome of the invocation. However, if the pre-filter raises the exception `CORBA::FILTER_SUPPRESS` no exception is returned to the caller—the

---

caller cannot tell that the operation call has not been processed as normal (the output and input/output parameters and the return value are those passed back by the pre-filter object).

**Notes** Orbix specific.

**See Also** CORBA::Object::\_getPre()  
CORBA::Object::\_attachPost()  
CORBA::Filter

### CORBA::Object::\_closeChannel()

**Synopsis** void CORBA::Object::\_closeChannel(  
CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());

**Description** Closes the underlying communications connection to the server. This function is intended as an optimization so that a connection between a client and server that is rarely used is not kept open for long periods between uses.

The channel is automatically reopened when an invocation is made on the object. If the client holds proxies for other objects in the same server, the channel is closed for all such proxies; it is automatically reopened when a subsequent invocation is made on one of these proxies.

**Notes** Orbix specific.

**See Also** CORBA::Object::\_hasValidOpenChannel()  
CORBA::ORB::closeChannel()

### CORBA::Object::\_create\_request()

**Synopsis** CORBA::ORBStatus CORBA::Object::\_create\_request(  
CORBA::Context\_ptr ctx,  
const char\* operation,  
CORBA::NVList\_ptr arg\_list,  
CORBA::NamedValue\_ptr& result,  
CORBA::Request\_ptr& request,  
CORBA::Flags req\_flags,  
CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());

**Description** Constructs a CORBA::Request object. Refer to CORBA::Object::\_request() for an alternative way to create a Request.

### Parameters

|                        |                                                                                                                                                                                                                                                                                                                                                |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ctx</code>       | Context object, if any, to be sent in the <code>Request</code> . If the <code>ctx</code> argument to <code>_create_request()</code> is a nil Context object reference, the Context may be added by calling the <code>ctx()</code> function on the <code>Request</code> object.                                                                 |
| <code>operation</code> | The name of the operation.                                                                                                                                                                                                                                                                                                                     |
| <code>arg_list</code>  | The parameters (each parameter in the list is of type <code>NamedValue</code> ). If the <code>arg_list</code> argument of the constructor is zero, you must add the arguments by calling the <code>arguments()</code> function on the <code>Request</code> object—one call to <code>arguments()</code> for each argument that is to be passed. |
| <code>result</code>    | Contains the return value of the operation.                                                                                                                                                                                                                                                                                                    |
| <code>req_flags</code> | If the flag <code>CORBA::OUT_LIST_MEMORY</code> is set, the storage associated with <code>out</code> parameters is assumed to be supplied by the caller. Otherwise, storage associated with <code>out</code> parameters is dynamically allocated.                                                                                              |

**Return Value** Returns 1 (`TRUE`) if successful, 0 (`FALSE`) otherwise.

**Notes** CORBA compliant. This function is part of the Dynamic Invocation Interface.

**See Also**

- `CORBA::Object::_request()`
- `CORBA::Request`
- `CORBA::Context`
- `CORBA::Flags`
- `CORBA::Request::arguments()`
- `CORBA::NVList`
- `CORBA::NamedValue`

### **CORBA::Object::\_deref()**

**Synopsis**

```
virtual void* CORBA::Object::_deref();
```

**Description** When the BOAImpl approach is used, this function returns a pointer to the BOAImpl base class of the target object's implementation object.

In the TIE approach, two objects exist: the TIE object and the true object, and the pointer returned is that of the true object.

You can redefine this function in the implementation class to allow casting from an interface to an implementation class. C++ prevents a simple cast because virtual inheritance is used in the inheritance hierarchy in the BOAImpl approach (in the TIE approach, the cast down is illegal anyway, because an implementation class does not inherit from its IDL C++ class). For example:

```
// C++
// TIE Implementation.
// Account is the interface class,
// Account_i is the implementation class.
Account_ptr p;
Account_i* p_i = (Account_i*)p; // Illegal.
```

You may wish to use a function defined on an implementation class but not in the IDL interface. To do so requires a pointer to the implementation class. You can achieve the cast by redefining the function `_deref()` in the implementation class:

```
// C++
class Account_i : public virtual AccountBOAImpl {

 virtual void* _deref() { return this; }
};
```

You can then achieve the cast as follows:

```
// C++
Account_ptr p =;
Account_i* p_i = (Account_i*) Deref(p);
```

The `DEREF` macro calls the `_deref()` function. If `_deref()` is not defined by `Account_i`, then it inherits an implementation that returns a pointer to the `BOAImpl` object. (The `DEREF` macro on a TIE returns a pointer to the true object.)

You could remove the need for the cast by defining the extra functions as IDL operations in the IDL interface. However, this would make these operations available to remote processes, possibly against the requirements of the application. Also, some C++ functions cannot be translated simply into IDL.

#### Notes

Orbix specific.

### CORBA::Object::\_duplicate()

#### Synopsis

```
static CORBA::Object_ptr _duplicate(CORBA::Object_ptr obj,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

|                     |                                                      |
|---------------------|------------------------------------------------------|
| <b>Description</b>  | Increments the reference count of <code>obj</code> . |
| <b>Return Value</b> | Returns a reference to <code>self</code> .           |
| <b>Notes</b>        | CORBA compliant.                                     |
| <b>See Also</b>     | <code>CORBA::release()</code>                        |

### **CORBA::Object::\_enableInternalLock()**

**Synopsis**

```
CORBA::Boolean CORBA::Object::_enableInternalLock(
 CORBA::Boolean isEnabled
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description**

In Orbix, each `CORBA::Object` includes an internal read/write lock that is used by Orbix to synchronise concurrent access to the Orbix specific state of that object. A read lock is acquired, for example, if a thread calls the `CORBA::Object::_refCount()` member function. Similarly a write lock is acquired for the duration of the `_duplicate()` static member function on each IDL C++ class. However this read/write lock is not acquired when any application specific state of that object is accessed. For example, if an implementation class derives from a `BOAImpl` class that in turn derives (indirectly) from `CORBA::Object` adds member variables, or if a smart proxy does likewise, this additional state is not protected by the internal `CORBA::Object` read/write lock.

In principle, the internal `CORBA::Object` read/write lock could be made available to derived `BOAImpl` classes. In practice, however, there is a possibility that deadlock situations might occur because of interactions between Orbix's internal use of this lock, and the use made by you in a derived class. For this reason, access to the internal lock is discouraged.

Sometimes you may be certain that the Orbix specific state of a particular `CORBA::Object` instance will never be simultaneously accessed by different threads. This occurs, for example, if the instance is allocated automatically (on a thread stack), rather than on the heap; or if the semantics of the application are such that concurrent access can never occur. In these cases, the cost of acquiring and releasing the read/write lock in the Orbix member functions (such as `_duplicate()` and `_refCount()`) may be unwarranted. These costs can be controlled by calling `_enableInternalLock()`.

---

Disable the locking code in a specific `CORBA::Object` instance with extreme caution. Ensure that you release application-level locks if you wish to continue after an exception is raised.

**Parameters**

`isEnabled` If set to 1 (`TRUE`), internal locking is enabled; if set to 0 (`FALSE`), locking is disabled.

**Return Value** Returns the previous setting.

**Notes** Orbix specific.

**CORBA::Object::\_fd()****Synopsis**

```
CORBA::Object::int _fd(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;
```

**Description**

Finds the file descriptor associated with the TCP/IP connection to a target remote object. It is applicable only to proxy objects.

When using libraries or systems that depend on the TCP/IP `select()` call you may need to know which file descriptors are scanned by Orbix to detect incoming events: a common use is to merge use of Orbix with X-Windows event handling. The set of such file descriptors is returned by

```
CORBA::BOA::getFileDescriptors().
```

When used together with callbacks for connections, it allows you to work out, for example, when a connection to a particular remote object has been lost.

**Return Value** Returns the value `-1` if an error occurs or if the object is a real object rather than a proxy.

**Notes** Orbix specific.

**See Also** `CORBA::BOA::getFileDescriptors()`

**CORBA::Object::\_get\_implementation()****Synopsis**

```
CORBA::ImplementationDef_ptr CORBA::Object::_get_implementation(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Finds the name of the target object's server as registered in the Implementation Repository. For a local object in a server, this is that server's name if it is known, otherwise it is the process identifier of the server process. For an object created in the client, it is the process identifier of the client process. The server name is known if the server is launched by the Orbix daemon; or if it is launched manually and the server name is passed to `CORBA::BOA::impl_is_ready()` or set by `CORBA::ORB::setServerName()`.

**Notes** CORBA compliant.

**See Also** `CORBA::Object::_implementation()`

### **CORBA::Object::\_get\_interface()**

**Synopsis**

```
CORBA::InterfaceDef_ptr CORBA::Object::_get_interface(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Returns a reference to an object in the Interface Repository that describes the interface of this object.

**Notes** CORBA compliant.

**See Also** `CORBA::InterfaceDef`

### **CORBA::Object::\_getPost()**

**Synopsis**

```
void* CORBA::Object::_getPost() const;
```

**Description** Gets the object's post-filter.

**Return Value** Returns a pointer to the object's post-filter.

**Notes** Orbix specific.

**See Also** `CORBA::Object::_attachPost()`  
`CORBA::Object::_getPre()`  
`CORBA::Filter`

### **CORBA::Object::\_getPre()**

**Synopsis**

```
void* CORBA::Object::_getPre() const;
```



|                     |                                                                           |
|---------------------|---------------------------------------------------------------------------|
| <b>Description</b>  | Gets the object's pre-filter.                                             |
| <b>Return Value</b> | Returns a pointer to the object's pre-filter.                             |
| <b>Notes</b>        | Orbix specific.                                                           |
| <b>See Also</b>     | CORBA::Object::_attachPre()<br>CORBA::Object::_getPost()<br>CORBA::Filter |

### CORBA::Object::\_hash()

**Synopsis**      `CORBA::ULong CORBA::Object::_hash(CORBA::ULong maximum,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())`

**Description**      Every object reference has an internal identifier associated with it—a value that remains constant throughout the lifetime of the object reference.

The `_hash()` function returns a hashed value determined via a hashing function from the internal identifier. Two different object references may yield the same hashed value. However, if two object references return different hash values, these object references are known to be for different objects.

The `_hash()` function allows a developer to partition the space of object references into sub-spaces of potentially equivalent object references.

#### Parameters

`maximum`      The maximum value that is to be returned from the hash function. For example, setting `maximum` to 7 partitions the object reference space into a maximum of 8 sub-spaces (because the lower bound of the function is 0).

**Return Value**      A hashed value for the object reference in the range `0..maximum`.

**Notes**              CORBA compliant.

### CORBA::Object::\_hasValidOpenChannel()

**Synopsis**      `CORBA::Boolean CORBA::Object::_hasValidOpenChannel(  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())`

- Description** Determines whether the communications channel between the client and the server is open.
- This channel can be closed to avoid having an unnecessary connection left open for long periods between an idle client and server. The channel is automatically reopened when an invocation is made on the object.
- Notes** Orbix specific.
- See Also** `CORBA::Object::_closeChannel()`

### **CORBA::Object::\_host()**

- Synopsis**
- ```
const char* CORBA::Object::_host(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()  
const;
```
- Description** Returns the name of the host on which the target object is located.
- Notes** Orbix specific.

CORBA::Object::_implementation()

- Synopsis**
- ```
const char* CORBA::Object::_implementation(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()
const;
```
- Description** Finds the name of the target object's server as registered in the Implementation Repository. For a local object in a server, this is that server's name (if that server's name is known), otherwise it is the process' identifier. The server name is known if the server is launched by Orbix; or if it is launched manually and the server name is passed to `CORBA::BOA::impl_is_ready()` or `CORBA::BOA::obj_is_ready()` or set by `CORBA::ORB::setServerName()`.
- Notes** Orbix specific. The CORBA compliant version of this function is `CORBA::Object::_get_implementation()`.
- See Also** `CORBA::Object::_get_implementation()`

**CORBA::Object::\_interfaceHost()**

- Synopsis** `const char* CORBA::Object::_interfaceHost(  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())  
const;`
- Description** Finds the name of a host running an Interface Repository server that stores the target object's IDL definition.
- Notes** Orbix specific.

**CORBA::Object::\_interfaceImplementation()**

- Synopsis** `const char* CORBA::Object::_interfaceImplementation(  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())  
const;`
- Description** Returns the name of the Interface Repository server.
- Notes** Orbix specific.

**CORBA::Object::\_interfaceMarker()**

- Synopsis** `const char* CORBA::Object::_interfaceMarker(  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())  
const;`
- Description** Returns the name of the target object's interface.
- Notes** Orbix specific.

**CORBA::Object::\_is\_a()**

- Synopsis** `CORBA::Boolean CORBA::Object::_is_a(  
const char* logical_type_id,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Determines if a target object is an instance of the type specified in `logical_type_id` or is an instance of a derived type of the type in `logical_type_id`.

### Parameters

`logical_type_id` The fully scoped name of the IDL interface. You must use an underscore (‘\_’) rather than a scope operator (‘::’) to delimit scope in a name.

### Return Value

1 (TRUE) The object is an instance of the type specified by `logical_type_id` or if the type of the object is an instance of a derived type of that type.

0 (FALSE) The object is neither an instance of the type or a derived type of the type specified.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::useRemoteIsACalls()`  
`CORBA::ORB::usingRemoteIsACalls()`  
`CORBA::ORB::resortToStatic(CORBA::Boolean value)`

## **CORBA::Object::\_is\_equivalent()**

### Synopsis

```
CORBA::Boolean CORBA::Object::_is_equivalent(
 const CORBA::Object_ptr obj,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

### Description

Tests if two object references are equivalent. Two objects are equivalent if they have the same object reference, or they both refer to the same object.

### Parameters

`obj` The object to be compared for equivalence with the target object.

### Return Value

Returns 1 (TRUE) if the object references definitely refer to the same object. A return value of 0 (FALSE) does not necessarily mean that the object references are not equivalent—only that the ORB cannot confirm that they reference the same object.

**Notes** CORBA compliant.

**See Also** CORBA::Object::\_is\_a()

### CORBA::Object::\_isNull()

**Synopsis** CORBA::Boolean CORBA::Object::\_isNull(  
CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv())  
const;

**Description** If an object is created with an invalid object reference, for example by passing an invalid object reference string to the constructor:

```
Object(const char* obj_ref_string);
```

a null object is created. This function determines whether the invoked object is a null object.

**Return Value** Returns 1 (TRUE) if the object is a null object, 0 (FALSE) otherwise. This function also returns TRUE if the invoked object is a null proxy.

**Notes** Orbix specific.

**See Also** CORBA::Object::\_isNullProxy()  
CORBA::is\_nil()

### CORBA::Object::\_isNullProxy()

**Synopsis** CORBA::Boolean CORBA::Object::\_isNullProxy(  
CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv())  
const;

**Description** Tests if the object on which it is invoked is a null proxy. A null proxy is a proxy whose member functions propagate any exception passed to them. That is, if the Environment passed to an operation on a null proxy already references an exception, the function performs no action and returns the same Environment. For example, in the following code:

```
// C++
try {
 pPtr = <some operation call>;
 pPtr->op(IT_X);
}
catch(...){
```

```
...
}
```

`pPtr` may reference a null proxy if the operation call raises an exception.

**Return Value** Returns 1 (TRUE) value if the object is a null proxy and returns 0 (FALSE) otherwise.

**Notes** Orbix specific.

**See Also** `CORBA::is_nil()`

### **CORBA::Object::\_isRemote()**

**Synopsis**

```
virtual CORBA::Boolean CORBA::Object::_isRemote(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;
```

**Description** Determines whether or not an object reference is remote—that is, whether or not the object it references is in a different address space on the local or a remote host. It is virtual because it has a different implementation depending on whether the TIE or BOAImpl approach is used.

**Return Value** Returns 1 (TRUE) if the reference is to a remote object (that is, the target of the call is a proxy), returns 0 (FALSE) otherwise.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::collocated()`

### **CORBA::Object::\_loader()**

**Synopsis**

```
virtual CORBA::LoaderClass_ptr CORBA::Object::_loader(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;
```

**Description** Finds the target object's loader. Orbix provides a default loader for every object. This function must be called on a real object in a server; calling it from a proxy has no effect.

**Return Value** Returns a pointer to the object's loader (which may be the default loader if no user-defined loader is provided).

**Notes** Orbix specific.  
**See Also** CORBA::LoaderClass

### CORBA::Object::\_marker()

**Synopsis**

```
const char* CORBA::Object::_marker(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;
```

**Description** Finds the target object's marker name.

**Notes** Orbix specific.

**See Also** CORBA::Object::\_marker(const char\* marker)

### CORBA::Object::\_marker()

**Synopsis**

```
CORBA::Boolean CORBA::Object::_marker(const char* marker,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Sets the target object's marker name.

If the chosen marker is already in use for an object with the same interface within the server, Orbix silently assigns a different marker to the object (and the other object with the original marker is unaffected).

Use this function with care. Every incoming request to a server is dispatched to the appropriate object within the server on the basis of the marker (automatically) included in the request. Thus if an object is made known to a remote client (via `_bind()`, or as a return value or an `out/inout` parameter of an operation), and the object's marker is subsequently altered within the server by calling `_marker(const char*)`, a subsequent request from the remote client will fail because the client will have used the original value of the marker. Thus you should change the marker name of an object before knowledge of the existence of the object is passed from the server to any client.

See `CORBA::LoaderClass::rename()` for details of the algorithm executed by Orbix when `CORBA::Object::_marker()` is called.

**Return Value** Returns 1 (TRUE) if successful; returns 0 (FALSE) if marker is already in use within the process.

**Notes** Orbix specific.

**See Also** `CORBA::Object::_marker()`  
`CORBA::LoaderClass::rename()`

### **CORBA::Object::\_nil()**

**Synopsis**

```
static CORBA::Object_ptr CORBA::Object::_nil(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Returns a nil object reference for an Object.

**Notes** CORBA compliant.

**See Also** `CORBA::is_nil()`  
`CORBA::Object::_isNullProxy()`  
`CORBA::Object::_isNull()`

### **CORBA::Object::\_non\_existent()**

**Synopsis**

```
CORBA::Boolean CORBA::Object::_non_existent(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Tests if the target object exists. Normally this function is invoked on a proxy and determines whether the real object still exists.

**Return Value** Returns 1 (TRUE) if the object does not exist; returns 0 (FALSE) otherwise. (It does not raise an exception if the object does not exist.)

**Notes** CORBA compliant.

### **CORBA::Object::\_object\_to\_string()**

**Synopsis**

```
CORBA::Identifier CORBA::Object::_object_to_string(
 unsigned long kind = IT_ORBIX_OR_KIND,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Converts the target object's reference to a string. An stringified IOR object reference has the form:  
  
IOR: 00000000000000154c5d.....



- Return Value** Returns a stringified object reference. The caller is responsible for deleting the string returned.
- Notes** Orbix specific. See `CORBA::ORB::object_to_string()` for CORBA compliant version of this function.
- See Also** `CORBA::ORB::object_to_string()`

### **CORBA::Object::\_refCount()**

- Synopsis**  

```
CORBA::ULong CORBA::Object::_refCount(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()
 const;
```
- Description** Finds the target object's current reference count.
- Return Value** If called on a proxy `_refCount()` returns the reference count of the proxy object. If called on the true object in the server, it returns the reference count of the server object.
- Notes** Orbix specific.
- See Also** `CORBA::Object::_duplicate()`  
`CORBA::release()`

### **CORBA::Object::\_request()**

- Synopsis**  

```
CORBA::Request_ptr CORBA::Object::_request(
 CORBA::Identifier operation,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```
- Description** Constructs a `CORBA::Request` on the target object. This is the shorter form of `CORBA::Object::_create_request()`.  
  
You can add arguments and contexts after construction using `CORBA::Request::arguments()` and `CORBA::Request::ctx()`.
- Parameters**  
  
`operation` The name of the operation.
- Notes** CORBA compliant.

**See Also**      `CORBA::Object::_create_request()`  
                 `CORBA::Request::arguments()`  
                 `CORBA::Request::ctx()`

### **CORBA::Object::\_save()**

**Synopsis**      `virtual void CORBA::Object::_save(  
                  CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description**      Calls the `CORBA::LoaderClass::save()` function on the target object's loader. You must call the function `_save()` in the same address space as the target object: calling it in a client process—that is, on a proxy—has no effect.

**Notes**            Orbix specific.

**See Also**        `CORBA::LoaderClass`

# CORBA::ORB

## Synopsis

Class `CORBA::ORB` implements the OMG CORBA ORB pseudo interface and adds a number of functions specific to Orbix.

`CORBA::ORB` provides a set of functions that control Orbix from both the client and the server. These include operations to convert between strings and object references, and operations for use with the Dynamic Invocation Interface (DII). Additional Orbix specific functions allow clients have control over timeout duration, collocation control, assistance with interface matching, diagnostic levels and so on.

The functions on this class are invoked through the `CORBA::Orbix` object. In the client, this is a static object of class `CORBA::ORB`. On the server, it is a static object of class `CORBA::BOA`—a derived class of `CORBA::ORB`.

## CORBA

```
// Pseudo IDL
// In module CORBA
pseudo interface ORB (
 typedef sequence<Request> RequestSeq;
 typedef string OAid;

 BOA BOA_init(inout arg_list argv,
 in OAid boa_identifler);

 string object_to_string(in Object obj);
 Object string_to_object(in string str);

 Status create_list(in long count,
 out NVList new_list);
 Status create_operation_list(
 in OperationDef oper, out NVList new_list);
 Status create_named_value(
 out NamedValue nmval);

 Status get_default_context(out Context ctx);

 Status create_environment(
 out Environment new_env);
```

```
Status send_multiple_requests_oneway(
 in RequestSeq req);
Status send_multiple_requests_deferred(
 in RequestSeq req);

boolean poll_next_response();
Status get_next_response(out Request req);
};
```

### Orbix

```
// C++
typedef void(*OrbixIOCallback) (int);

typedef
enum { FD_OPEN_CALLBACK,
 FD_CLOSE_CALLBACK,
 FD_LOW_CALLBACK,
 FD_STOPLISTEN_CALLBACK,
 FD_STARTLISTEN_CALLBACK } OrbixIOCallbackType;

class ORB : public IT_PseudoIDL {

public:

 ORB();

 virtual ~ORB();

 Boolean pingDuringBind() const;

 Boolean resortToStatic() const;

 ULong defaultTxTimeout() const;

 ULong defaultTxTimeout(
 ULong val,
 Environment& IT_env =
 IT_chooseDefaultEnv());

 ULong connectionTimeout() const;

 ULong maxConnectRetries() const;

 Boolean mustRedefineDeref() const;
```

---

```
ULong defaultRxTimeout() const;

Boolean isDefaultRxTimeoutSet() const;

Boolean supportBidirectionalIIOP() const;

Object_ptr string_to_object(
 const char* ior,
 Environment& IT_env =
 IT_chooseDefaultEnv());

string object_to_string(
 Object_ptr obj,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Status create_list(Long count,
 NVList_ptr& new_list,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Status create_named_value(
 NamedValue_ptr& rNamedValue,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Status get_default_context(
 Context_ptr& context,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Status create_operation_list(
 OperationDef_ptr operation,
 NVList_ptr& list,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Status create_environment(
 Environment_ptr& new_env,
 Environment& IT_env =
 IT_chooseDefaultEnv());
```

```
typedef <sequence>Request RequestSeq;

Status send_multiple_requests_oneway(
 const RequestSeq& requests,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Status send_multiple_requests_deferred(
 const RequestSeq& requests,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Status get_next_response(
 Request_ptr& req,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean poll_next_response(
 Environment& IT_env =
 IT_chooseDefaultEnv());

void useRemoteIsACalls(
 Boolean useRemoteCall,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean usingRemoteIsACalls(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

void useServiceContexts(
 Boolean useServiceContexts,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean usingServiceContexts(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

Boolean registerPerRequestServiceContextHandler(
 ServiceContextHandler* CtxHandler,
 Environment& IT_env =
 IT_chooseDefaultEnv());
```

---

```
Boolean registerPerObjectServiceContextHandler(
 ServiceContextHandler* CtxHandler,
 Object* theObject,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean unregisterPerRequestServiceContextHandler(
 ULong& CtxId,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean unregisterPerObjectServiceContextHandler(
 ULong& CtxId,
 Object* theObject,
 Environment& IT_env =
 IT_chooseDefaultEnv());

BOA_ptr BOA_init(
 int& argc,
 char** argv,
 const char* oa_identifier,
 Environment& IT_env =
 IT_chooseDefaultEnv());

static void useHostNameInIOR(
 Boolean val,
 Environment& IT_env =
 IT_chooseDefaultEnv());

static void useReverseLookup(Boolean enable);

static Boolean usingReverseLookup();

typedef _IDL_SEQUENCE_string* ObjectIdList_ptr;

ObjectIdList_ptr list_initial_services(
 Environment& IT_env =
 IT_chooseDefaultEnv());

Object_ptr resolve_initial_references(
 const char* identifier,
 Environment& IT_env =
```

```
 IT_chooseDefaultEnv());

_IDL_SEQUENCE_octet* makeOrbixObjectKey(
 const char* host,
 const char* serverName,
 const char* interfaceName,
 Environment& IT_env =
 IT_chooseDefaultEnv());

typedef _IDL_SEQUENCE_octet* ObjectKey;

string makeIOR(
 const char* host,
 unsigned short port,
 ObjectKey objKey,
 const char* typeID,
 Environment& IT_env =
 IT_chooseDefaultEnv());

ObjectRef string_to_object(
 const char* host,
 const char* impl,
 const char* marker,
 const char* intfHost,
 const char* intfImpl,
 const char* intfMarker,
 Environment& IT_env =
 IT_chooseDefaultEnv());

OrbixIOCallback registerIOCallback(
 OrbixIOCallback IOcallback,
 OrbixIOCallbackType IOcallbackType);

Boolean useTransientPort() const;

Boolean useTransientPort(
 Boolean turnOn);

ULong defaultRxTimeout(
 ULong val,
 Environment& IT_env =
 IT_chooseDefaultEnv());
```



---

```
Boolean collocated(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

Boolean collocated(
 Boolean turnOn,
 Environment& IT_env =
 IT_chooseDefaultEnv());

const char* myHost(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

const char* myServer(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

void reSizeObjectTable(
 ULong newSize,
 Environment& IT_env =
 IT_chooseDefaultEnv());

void setServerName(
 const char* serverName,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean isBaseInterfaceOf(
 const char* derived,
 const char* maybeABase,
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

void baseInterfacesOf(
 _IDL_SEQUENCE_string& interfaces,
 const char* derived,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Short setDiagnostics(
 Short level,
 Environment& IT_env =
 IT_chooseDefaultEnv());
```

```
Short getDiagnostics() const;

Boolean pingDuringBind(
 Boolean pingOn,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean optimiseProtocolEncoding() const;

Boolean optimiseProtocolEncoding(
 Boolean value,
 Environment& IT_env =
 IT_chooseDefaultEnv());

ULong connectionTimeout(
 ULong timeout,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean abortSlowConnects(
 Boolean state);

Boolean abortSlowConnects() const;

Boolean eagerListeners(
 Boolean newPrio,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean eagerListeners(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

ULong maxConnectRetries(
 ULong retries,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean noReconnectOnFailure() const;

Boolean noReconnectOnFailure(
 Boolean dont_reconnect,
```

---

```
Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean mustRedefineDeref(
 Boolean onOff,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean supportBidirectionalIIOP(
 Boolean on,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean resortToStatic(
 Boolean value,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean closeChannel(
 int fd,
 Environment& IT_env =
 IT_chooseDefaultEnv());

IT_PFV set_unsafeDelete(IT_PFV pfv);

IT_PFB set_unsafeFDClose(IT_PFB fn);

IT_reqTransformer* setMyReqTransformer(
 IT_reqTransformer* transformer,
 Environment& IT_env =
 IT_chooseDefaultEnv());

IT_reqTransformer* getMyReqTransformer() const;

void setReqTransformer(
 IT_reqTransformer* transformer,
 const char* server,
 const char* host = 0,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Boolean bindUsingIIOP() const;
```

```
Boolean bindUsingIIOP(Boolean on);

static ORB_ptr IT_create(
 Environment& IT_env =
 IT_chooseDefaultEnv());

static ORB_ptr _duplicate(
 ORB_ptr obj,
 Environment& IT_env =
 IT_chooseDefaultEnv());

static ORB_ptr _nil(
 Environment& IT_env =
 IT_chooseDefaultEnv());

static Boolean GetConfigValue(
 const char* name,
 char*& value);

static Boolean SetConfigValue(
 const char* name,
 char* value);

static Boolean GetConfigValueLong(
 const char* name,
 Long& value);

static Boolean SetConfigValueLong(
 const char* name,
 Long value);

static Boolean GetConfigValueBool(
 const char* name,
 Boolean& value);

static Boolean SetConfigValueBool(
 const char* name,
 Boolean value);

static void PlaceCVHandlerBefore(
 const char* handler,
 const char* beforeThisHandler);
```

---

```
static void PlaceCVHandlerAfter(
 const char* after,
 const char* before);

static void ActivateCVHandler(
 const char* identifier);

static void DeactivateCVHandler(
 const char* identifier);

static void ReinitialiseConfig();

static void Output(const char* string,
 int level = 1);

static void Output(Environment& e,
 int level = 1);

static void Output(SystemException* x,
 int level = 1);

static void ActivateOutputHandler(
 const char* identifier);

static void DeactivateOutputHandler(
 const char* identifier);

Boolean registerIOCallbackObject(
 unsigned char eventType,
 IT_IOCallback* obj);

Boolean unregisterIOCallbackObject(
 unsigned char eventType,
 IT_IOCallback* obj);

void addForeignFDSet(fd_set& theFDset,
 unsigned char aState);

void removeForeignFDSet(fd_set& theFDset,
 unsigned char aState);

void addForeignFD(const int fd,
 unsigned char aState);
```

```
void removeForeignFD(const int fd,
 unsigned char aState);

fd_set getForeignFDSet() const;

fd_set getSelectableFDSet() const;

fd_set getAllOrbixFDs() const;

Boolean isOrbixFD(int fd);

Boolean isForeignFD(
 int fd,
 unsigned char mask = FD_FOREIGN_WRITE |
 FD_FOREIGN_READ |
 FD_FOREIGN_EXCEPT);

Boolean isOrbixSelectableFD(int fd);

Boolean add_nw_threads(ULong num_threads);

};
```

**CORBA::ORB::abortSlowConnects()****Synopsis**

```
CORBA::Boolean CORBA::ORB::abortSlowConnects (
 CORBA::Boolean value);
```

**Description**

In cases where a node is known to the local node, but down or unreachable, a TCP/IP connect can block for a considerable time. If `value` is set to 1 (`TRUE`), `abortSlowConnects()` aborts TCP/IP connection attempts which exceed the time-out specified in `CORBA::ORB::connectionTimeout()`. The default value for this time-out is 30 seconds.

**Parameters**

`value`      The Boolean value for `abortSlowConnects` flag. 1 (`TRUE`) to force slow TCP/IP connections to be aborted, 0 (`FALSE`) to continue to wait.

**Return Value**      Returns the previous setting.

**Notes**              Orbix specific.

**See Also**

```
CORBA::ORB::connectionTimeout ()
CORBA::ORB::connectionTimeout (CORBA::Boolean)
CORBA::ORB::eagerListeners ()
```

**CORBA::ORB::abortSlowConnects()****Synopsis**

```
CORBA::Boolean CORBA::ORB::abortSlowConnects () const;
```

**Description**

Returns the current setting of the `abortSlowConnects` flag.

**Return Value**

|                          |                                                                                                                      |
|--------------------------|----------------------------------------------------------------------------------------------------------------------|
| 1 ( <code>TRUE</code> )  | Abort TCP/IP connection attempts that exceed the timeout specified in <code>CORBA::ORB::connectionTimeout()</code> . |
| 0 ( <code>FALSE</code> ) | Slow connection attempts are not aborted. This is the default value.                                                 |

**Notes**              Orbix specific.

**See Also**

```
CORBA::ORB::connectionTimeout (CORBA::ULong)
CORBA::ORB::eagerListeners (CORBA::Boolean)
CORBA::ORB::abortSlowConnects (CORBA::Boolean)
```

### **CORBA::ORB::ActivateCVHandler()**

- Synopsis**      `static void CORBA::ORB::ActivateCVHandler(  
                  const char* identifier);`
- Description**    Activates the configuration value handler identified in `identifier`.  
  
The function `ReinitialiseConfig()` must be called before modifications by this function take effect.
- Notes**          Orbix specific.
- See Also**        `CORBA::ORB::ReinitialiseConfig()`  
`CORBA::ORB::DeactivateCVHandler()`  
`CORBA::UserCVHandler`  
`CORBA::ExtraRegistryCVHandler`  
`CORBA::ExtraConfigFileHandler`

### **CORBA::ORB::ActivateOutputHandler()**

- Synopsis**      `static void CORBA::ORB::ActivateOutputHandler(  
                  const char* identifier);`
- Description**    Activates the output handler specified in `identifier`.
- Notes**          Orbix specific.
- See Also**        `CORBA::UserOutput`

### **CORBA::ORB::addForeignFD()**

- Synopsis**      `void CORBA::ORB::addForeignFD(  
                  const int fd,  
                  unsigned char aState);`
- Description**    Orbix allows you to add foreign file descriptors to the Orbix event loop. Orbix then monitors these file descriptors when you call an Orbix event processing function, such as `CORBA::ORB::processEvents()`. To receive callbacks on those foreign file descriptors, you must implement a class that inherits from `CORBA::IT_IOCallback`.  
  
The function `CORBA::ORB::addForeignFD()` allows you to add a foreign file descriptor to the Orbix event processing loop.



**Parameters**

|                     |                                                                                                                                                                                                                                           |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>fd</code>     | The file descriptor to be added to the Orbix event processing loop.                                                                                                                                                                       |
| <code>aState</code> | Indicates the type of events for which the file descriptor should be monitored. This can be <code>FD_FOREIGN_READ</code> , <code>FD_FOREIGN_WRITE</code> , and <code>FD_FOREIGN_EXCEPT</code> , or a logical combination of these values. |

**Notes** Orbix specific.

**See Also** `CORBA::ORB::addForeignFDSet()`  
`CORBA::ORB::removeForeignFD()`  
`CORBA::ORB::removeForeignFDSet()`

**CORBA::ORB::addForeignFDSet()****Synopsis**

```
void CORBA::ORB::addForeignFDSet(
 fd_set& fds, unsigned char aState);
```

**Description**

Orbix allows you to add foreign file descriptors to the Orbix event loop. Orbix then monitors these file descriptors when you call an Orbix event processing function, such as `CORBA::ORB::processEvents()`. To receive callbacks on those foreign file descriptors, you must implement a class that inherits from `CORBA::IT_IOCallback`.

The function `CORBA::ORB::addForeignFDSet()` allows you to add a set of foreign file descriptors to the Orbix event processing loop.

**CORBA::ORB::baseInterfacesOf()****Synopsis**

```
void CORBA::ORB::baseInterfacesOf(
 _IDL_SEQUENCE_string& interfaces,
 const char* derived,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description**

Returns an IDL sequence of strings in the parameter `interfaces`, listing the base interfaces of `derived`. The interface `derived` is returned in the sequence, since it is considered a base interface of itself.

### Parameters

`interfaces` The caller is responsible for deleting the returned IDL sequence.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::isBaseInterfaceOf()`

### **CORBA::ORB::bindUsingIIOP()**

#### Synopsis

```
CORBA::Boolean CORBA::ORB::bindUsingIIOP(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()
 const;
```

#### Description

Orbix clients can call the function `_bind()` to obtain a reference to a distributed object. By default, `_bind()` uses the CORBA Internet Inter-ORB Protocol (IIOP) when attempting to return an object reference. This function indicates whether `_bind()` currently uses IIOP or the non-standard Orbix communications protocol.

**Return Value** Returns a 1 (TRUE) if `_bind()` currently uses IIOP. Returns 0 (FALSE) otherwise.

**Notes** Orbix specific.

### **CORBA::ORB::bindUsingIIOP()**

#### Synopsis

```
CORBA::Boolean CORBA::ORB::bindUsingIIOP(
 CORBA::Boolean on,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

#### Description

This function allows you to specify whether the Orbix `_bind()` function should use the CORBA Internet Inter-ORB Protocol (IIOP) or the non-standard Orbix communications protocol. By default, `_bind()` uses IIOP.

---

**Note:** The `bindUsingIIOP()` function allows you to set at start of program the protocol used by `_bind()`. If you use `bindUsingIIOP()` to switch from the Orbix protocol to IIOP between successive binds to the same object, it has no effect.

---

**Parameters**

on 1 (TRUE) specifies that `_bind()` should use IIOP. 0 (FALSE) specifies that `_bind()` should use the Orbix protocol.

**Return Value** Returns the previous setting.

**Notes** Orbix specific.

**CORBA::ORB::BOA\_init()****Synopsis**

```
CORBA::BOA_ptr CORBA::ORB::BOA_init(
 int& argc,
 char** argv,
 const char** oa_identifier,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description**

Initializes a server's connection to the Basic Object Adapter (BOA). A compliant program first obtains a pointer to the ORB using `CORBA::ORB_init()` as follows:

```
// C++
CORBA::ORB_ptr orb =
 CORBA::ORB_init(argc, argv, "Orbix");
CORBA::BOA_ptr boa =
 orb->BOA_init(argc, argv, "Orbix_BOA");
```

In Orbix, the object reference returned by these functions is a reference to the `CORBA::Orbix` object.

**Parameters**

|      |                                                                                                                                                                                                                                                                                                                                                                                                        |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| argc | The number of arguments in argv.                                                                                                                                                                                                                                                                                                                                                                       |
| argv | A sequence of option or configuration strings that are used if <code>oa_identifier</code> is a null string. Each string is of the form:<br>-OA<suffix> <value><br>where <suffix> is the name of the option being set, and <value> is the value to which the option is to be set. Any string not in this format is ignored. An example parameter to identify the Orbix ORB's BOA is:<br>-OAid Orbix_BOA |

`oa_identifier` A string identifying the object adapter. The string "Orbix\_BOA" identifies the Orbix ORB's Basic Object Adapter.

If this parameter is null, the content of `argv` is checked.

**Return Value** An object reference to a Basic Object Adapter. In Orbix, it is not necessary to call this function before using the ORB since Orbix automatically initializes a client or server's connection, making access to the ORB available through the `CORBA::Orbix` object.

**Notes** CORBA compliant.

**See Also** `CORBA::ORB_init()`

### **CORBA::ORB::closeChannel()**

**Synopsis**

```
CORBA::Boolean CORBA::ORB::closeChannel(int fd,
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Requests Orbix to close the communications channel to the server. This function is intended as an optimization so that a connection between an idle client and server is not kept open for long periods between uses.

The channel is automatically reopened when an invocation is made on an object in the server.

#### **Parameters**

`fd` The file descriptor identifying the channel.

**Notes** Orbix specific.

**See Also** `CORBA::Object::_hasValidOpenChannel()`  
`CORBA::Object::_closeChannel()`  
`CORBA::Object::_fd()`

### **CORBA::ORB::collocated()**

**Synopsis**

```
CORBA::Boolean CORBA::ORB::collocated(
```

---

```

CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()
const;

```

**Description** Determines whether collocation is set. Binding to objects outside of the current process' address space is prevented if collocation is set. If collocation is not set, the lookup mechanism allows binding outside of the current address space. By default, collocation is not set.

---

**Note:** If you call `collocated(1)` before calling `impl_is_ready()`, the `impl_is_ready()` call returns immediately without contacting the Orbix daemon.

---

**Return Value** Returns 1 (TRUE) if collocation is set and returns 0 (FALSE) otherwise.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::collocated(CORBA::Boolean turnOn)`  
`CORBA::Object::_isRemote()`

### **CORBA::ORB::collocated()**

**Synopsis**

```

CORBA::Boolean CORBA::ORB::collocated(
 CORBA::Boolean turnOn,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

```

**Description** Enforces collocation if `turnOn` is set to 1 (TRUE). If collocation is enforced, binding to objects outside of the current process' address space is prevented. If set to 0 (FALSE), the lookup mechanism allows binding outside of the current address space. By default, collocation is not set.

**Return Value** Returns the previous setting.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::collocated()`  
`CORBA::Object::_isRemote()`

### **CORBA::ORB::connectionTimeout()**

- Synopsis** `CORBA::ULong CORBA::ORB::connectionTimeout (`  
    `CORBA::ULong timeout,`  
    `CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Sets the time limit, in seconds, for establishing that a connection from a client to a server is fully operational. The default is 30 seconds, which should be adequate in the majority of cases.
- The value set by this function comes into effect if, for example, the server crashes after the transport (for example, TCP/IP) connection has been made but before the full Orbix connection has been established.
- The value set by `connectionTimeout()` is independently used by `abortSlowConnects()` when setting up the transport connection.
- If clients of a single-threaded server continually time out because the server is busy, it may be that existing connections are being favoured over new connection attempts. The function `CORBA::ORB::eagerListeners()` addresses this problem.
- Parameters**
- |                      |                                 |
|----------------------|---------------------------------|
| <code>timeout</code> | The desired timeout in seconds. |
|----------------------|---------------------------------|
- Return Value** Returns the previous setting.
- Notes** Orbix specific.
- See Also** `CORBA::ORB::abortSlowConnects()`  
`CORBA::ORB::maxConnectRetries()`  
`CORBA::Environment::timeout()`  
`CORBA::ORB::eagerListeners()`

### **CORBA::ORB::connectionTimeout()**

- Synopsis** `CORBA::ULong CORBA::ORB::connectionTimeout() const;`
- Description** Returns the time limit, in seconds, allowed to establish that a connection from a client to a server is fully operational.
- Return Value** The current value, in seconds, of the connection timeout.
- Notes** Orbix specific.

**See Also** CORBA::ORB::connectionTimeout (CORBA::ULong)  
CORBA::ORB::abortSlowConnects (CORBA::Boolean)

### **CORBA::ORB::create\_environment()**

**Synopsis** CORBA::Status CORBA::ORB::create\_environment(  
CORBA::Environment\_ptr& new\_env,  
CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());

**Description** Returns a newly-created Environment in the parameter new\_env.

**Return Value** Returns 1 (TRUE) if successful, 0 (FALSE) otherwise.

**Notes** CORBA compliant.

**See Also** CORBA::Environment

### **CORBA::ORB::create\_list()**

**Synopsis** CORBA::Status CORBA::ORB::create\_list(  
CORBA::Long count,  
CORBA::NVList\_ptr& new\_list,  
CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());

**Description** Allocates space for an empty NVList, new\_list, of size count to contain NamedValue objects. You can use a NamedValue struct as a parameter type or as a way to describe arguments to a request when using the Dynamic Invocation Interface.

#### **Parameters**

count        Number of elements in the new NVList.  
new\_list     A pointer to the start of the list. The caller must release the reference when it is no longer needed, or assign it to a NVList\_var variable for automatic management.

**Return Value** Returns 1 (TRUE) if successful, 0 (FALSE) otherwise.

**Notes** CORBA compliant. This function is part of the Dynamic Invocation Interface.

**See Also** CORBA::NVList  
CORBA::ORB::create\_operation\_list()

CORBA::NamedValue  
CORBA::Request

### CORBA::ORB::create\_named\_value()

#### Synopsis

```
CORBA::Status CORBA::ORB::create_named_value(
 CORBA::NamedValue_ptr& rNamedValue,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

#### Description

Required for creating `NamedValue` objects to be used as return value parameter for the `Object::_create_request` operation when using the Dynamic Invocation Interface.

#### Parameters

`rNamedValue` A pointer to the `NamedValue`. The caller must release the reference when it is no longer needed, or assign it to a `NamedValue_var` variable for automatic management.

#### Return Value

Returns 1 (`TRUE`) if successful, 0 (`FALSE`) otherwise.

#### Notes

CORBA compliant. This function is part of the Dynamic Invocation Interface.

#### See Also

CORBA::NVList  
CORBA::Request  
CORBA::NamedValue

### CORBA::ORB::create\_operation\_list()

#### Synopsis

```
CORBA::Status CORBA::ORB::create_operation_list(
 CORBA::OperationDef_ptr operation,
 CORBA::NVList_ptr& list,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

#### Description

Returns an `NVList`, in the parameter `list`, initialized with the argument descriptions for the operation specified in `operation`. The returned `NVList` is of the correct length (with one element per argument), and each `NamedValue` element of the list has a valid name and valid flags (denoting the argument passing mode). The value (of type `CORBA::Any`) of the `NamedValue` has a valid type (denoting the type of the argument). The value of the argument is not filled in.



**Parameters**

|                        |                                                                                                                                                                                     |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>operation</code> | A reference to the Interface Repository object describing the operation.                                                                                                            |
| <code>list</code>      | A pointer to the start of the list. The caller must release the reference when it is no longer needed, or assign it to a <code>NVList_var</code> variable for automatic management. |

**Return Value** Returns 1 (TRUE) if successful, 0 (FALSE) otherwise.

**See Also** `CORBA::NVList`  
`CORBA::Any`  
`CORBA::create_list()`  
`CORBA::NamedValue`

**CORBA::ORB::DeactivateCVHandler()**

**Synopsis**

```
static void CORBA::ORB::DeactivateCVHandler(
 const char* identifier);
```

**Description** Deactivates the configuration value handler identified in `identifier`.  
The function `ReinitialiseConfig()` must be called before modifications by this function take effect.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::ActivateCVHandler()`  
`CORBA::ORB::ReinitialiseConfig()`  
`CORBA::UserCVHandler`  
`CORBA::ExtraRegistryCVHandler`  
`CORBA::ExtraConfigFileHandler`

**CORBA::ORB::DeactivateOutputHandler()**

**Synopsis**

```
static void CORBA::ORB::DeactivateOutputHandler(
 const char* identifier);
```

**Description** Deactivates the output handler specified in `identifier`.

**Notes** Orbix specific.

**See Also** `CORBA::UserOutput`

### **CORBA::ORB::DEFAULT\_TIMEOUT**

**Synopsis** `static const CORBA::ULong CORBA::ORB::DEFAULT_TIMEOUT;`

**Description** Defines the default number of milliseconds that a server should wait between events: a timeout occurs if Orbix has to wait longer than the timeout value for the next event. The default timeout is 60,000 milliseconds (1 minute).

**Notes** Orbix specific.

**See Also** `CORBA::ORB::INFINITE_TIMEOUT`  
`CORBA::ORB::defaultTxTimeout()`  
`CORBA::BOA::impl_is_ready()`  
`CORBA::BOA::processNextEvent()`  
`CORBA::BOA::processEvents()`

### **CORBA::ORB::defaultRxTimeout()**

**Synopsis** `CORBA::ULong CORBA::ORB::defaultRxTimeout(  
CORBA::ULong val,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description** Incoming IIOp messages may be fragmented. By default, the process will wait for each fragment of the incoming IIOp message to be received. The user may call this function to set a timeout for receiving each fragment of an incoming IIOp message.

By default RxTimeout is not set. As a consequence, a process will hang waiting for the next fragment of an incoming IIOp message to be received.

The supplied timeout is specified in milliseconds.

**Exceptions** If RxTimeout has been set and any fragment of an incoming IIOp message is not received within the specified timeout period, a `CORBA::COMM_FAILURE` exception will be raised.

**Return Value** Returns the previous setting, in milliseconds.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::defaultRxTimeout()`

---

```
CORBA::ORB::isDefaultTxTimeoutSet ()
```

### CORBA::ORB::defaultRxTimeout()

**Synopsis**

```
CORBA::ULong CORBA::ORB::defaultRxTimeout () const;
```

**Description**

Returns the current value of the timeout to be used to receive each fragment of an incoming IOP message.

The value returned by `CORBA::ORB::defaultRxTimeout ()` is valid if and only if the value returned by the associated accessor function `CORBA::ORB::isDefaultTxTimeoutSet ()` is 1 (TRUE).

**Return Value**

The current value of the timeout, in milliseconds.

**Exceptions**

If RxTimeout has been set and any fragment of an incoming IOP message is not received within the specified timeout period, a `CORBA::COMM_FAILURE` exception will be raised.

**Notes**

Orbix specific.

**See Also**

```
CORBA::ORB::defaultRxTimeout (CORBA::ULong)
CORBA::ORB::isDefaultTxTimeoutSet ()
```

### CORBA::ORB::defaultTxTimeout()

**Synopsis**

```
CORBA::ULong CORBA::ORB::defaultTxTimeout (
 CORBA::ULong val,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv ());
```

**Description**

Sets the timeout for all remote calls and returns the previous setting. If a timeout is set in an `Environment`, it supersedes any value set globally by this function. By default, no call has a timeout, that is, the default timeout is infinite.

The value set by this function is ignored when making a connection between a client and a server. It comes into effect only when this connection has been established. This timeout value may be overridden by the timeout values set in an `Environment` object by the use of the mutator

```
CORBA::Environment::timeout (CORBA::ULong t).
```

### Parameters

val                    The desired global transmit value in milliseconds.

**Return Value**      Returns the previous setting.

**Exceptions**        A subsequent remote call that does not return within the given timeout interval fails with a `CORBA::COMM_FAILURE` exception.

**Notes**             Orbix specific.

**See Also**          `CORBA::ORB::defaultTxTimeout()`  
`CORBA::Environment::timeout(CORBA::ULong)`  
`CORBA::Environment::timeout()`

### **CORBA::ORB::defaultTxTimeout()**

**Synopsis**            `CORBA::ULong CORBA::ORB::defaultTxTimeout() const;`

**Description**       Returns the current global timeout value for all remote calls between client and server. This timeout applies only after the connection between a client and server has been established.

The value returned is the global transmit timeout value as set by the function `CORBA::ORB::defaultTxTimeout(CORBA::ULong)`. This timeout value may be overridden by the timeout values set in an Environment object by the use of the mutator `CORBA::Environment::timeout(CORBA::ULong t)`.

The initial value for timeout in a new Environment object is 0.

**Return Value**      The current global transmit timeout value in seconds.

**Notes**             Orbix specific.

**See Also**          `CORBA::ORB::defaultTxTimeout(CORBA::ULong)`  
`CORBA::Environment::timeout(CORBA::ULong)`  
`CORBA::Environment::timeout()`

### **CORBA::ORB::eagerListeners()**

**Synopsis**            `CORBA::Boolean eagerListeners(  
                          CORBA::Boolean newPrio,  
                          CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description** By default, currently established connections to a server are given priority over requests for new connections. As a result, busy single-threaded servers (for example, processing long running operations) may not service new connection attempts and consequently clients attempting to make a connection may continually time out.

The function `eagerListeners()` allows equal fairness to be given to both established connections and to new connection attempts and so avoids discrimination against new connections resulting from the default behavior.

This feature is not necessary in multi-threaded versions of Orbix.

**Parameters**

`newPrio` A value of 1 (TRUE) enables eager listening—this means that attempts to establish new connections are given equal priority to processing of established connections; a value of 0 (FALSE) re-establishes the default behavior.

**Return Value** Returns the previous value.

**Notes** Orbix specific.

**See Also** `CORBA::Boolean eagerListeners(CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`  
`CORBA::ORB::connectionTimeout()`

## CORBA::ORB::eagerListeners()

**Synopsis** `CORBA::Boolean eagerListeners(CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description** Determines if eager listening is set. Refer to the overloaded modifier function for details.

**Notes** Orbix specific.

**See Also** `CORBA::Boolean eagerListeners(CORBA::Boolean value, CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

### **CORBA::ORB::getAllOrbixFDs()**

- Synopsis** `fd_set CORBA::ORB::getAllOrbixFDs() const;`
- Description** Returns the current set of all Orbix file descriptors. This may include file descriptors that are not returned by `CORBA::ORB::getSelectableFDSet()`.
- Return Value** Returns the complete file descriptor set.
- Notes** Orbix specific.
- See Also** `CORBA::getSelectableFDSet()`

### **CORBA::ORB::getForeignFDSet()**

- Synopsis** `fd_set CORBA::ORB::getForeignFDSet() const;`
- Description** Returns the set of all foreign file descriptors added to the Orbix event processing loop using `CORBA::ORB::addForeignFD()` or `addForeignFDSet()`.
- Return Value** Returns the current set of foreign file descriptors.
- Notes** Orbix specific.
- See Also** `CORBA::ORB::addForeignFD()`  
`CORBA::ORB::addForeignFDSet()`

### **CORBA::ORB::GetConfigValue()**

- Synopsis** `static CORBA::Boolean CORBA::ORB::GetConfigValue(  
    const char* name,  
    char*& value);`

- Description** Obtains the value of the configuration entry specified.

---

**Note:** A `CORBA::string_var` object may be used to store the value as an implicit type conversion to `char*&` has been defined

---

#### **Parameters**

`name` The name of the configuration item.

---

`value` A reference to a pointer to hold the value of the configuration item. The memory returned is to be owned by the caller and should be freed up with `delete []` after using it.

**Return Value** 1 (TRUE) if successful, 0 (FALSE) otherwise.

**Notes** Orbix specific.

### **CORBA::ORB::GetConfigValueBool()**

**Synopsis**

```
static CORBA::Boolean CORBA::ORB::GetConfigValueBool(
 const char* name,
 CORBA::Boolean& value);
```

**Description** Obtains the value of the configuration entry specified.

**Parameters**

`name` The name configuration item.  
`value` A reference to a bool variable to store the value in.

**Return Value** 1 (TRUE) if successful, 0 (FALSE) otherwise.

**Notes** Orbix specific.

### **CORBA::ORB::GetConfigValueLong()**

**Synopsis**

```
static CORBA::Boolean GetConfigValueLong(
 const char* name,
 CORBA::Long& value);
```

**Description** Obtains the value of the configuration entry specified.

**Parameters**

`name` The name configuration item.  
`value` A reference to a long variable to store the value in.

**Return Value** 1 (TRUE) if successful, 0 (FALSE) otherwise.

**Notes** Orbix specific.

### **CORBA::ORB::getDiagnostics()**

**Synopsis** `CORBA::Short CORBA::ORB::getDiagnostics() const;`

**Description** Returns the current level of diagnostic messages output by the Orbix libraries.

**Return Value**

- 0 No diagnostics.
- 1 Simple diagnostics. This is the default value.
- 2 Full diagnostics.
- 3 Fulll diagnostics with additional IOP tracing.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::setDiagnostics()`

### **CORBA::ORB::getMyReqTransformer()**

**Synopsis** `CORBA::IT_reqTransformer*  
CORBA::IT_reqTransformer::getMyReqTransformer();`

**Description** Returns the registered transformer object; returns 0 if no transformer registered.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::setMyReqTransformer()`

### **CORBA::ORB::getSelectableFDSet()**

**Synopsis** `fd_set CORBA::ORB::getSelectableFDSet() const;`

**Description** Returns the set of Orbix file descriptors that are guaranteed to be active when Orbix has events to process. This may be a subset of the complete Orbix file descriptor set.



---

An application must call `getSelectableFDSet()` during its initialization, in order to instruct Orbix to initialize for the file descriptor set.

**Return Value** Returns the active file descriptor set.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::getAllOrbixFDs()`

### **CORBA::ORB::get\_default\_context()**

**Synopsis**

```
CORBA::Status CORBA::ORB::get_default_context(
 CORBA::Context_ptr& context,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Returns a `CORBA::Context` object, in the parameter `context`, representing the process' default context. Refer to `CORBA::Context` for an explanation of the default context.

**Return Value** Returns 1 (TRUE) if successful, 0 (FALSE) otherwise.

**Notes** CORBA compliant.

**See Also** `CORBA::NVList`

### **CORBA::ORB::get\_next\_response()**

**Synopsis**

```
CORBA::Status CORBA::ORB::get_next_response(
 CORBA::Request_ptr& req,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** May be called successively to determine the outcomes of the individual requests specified in a `CORBA::send_multiple_requests()` call. The order in which responses are returned is not necessarily related to the order in which the requests are completed.

#### **Parameters**

`req` An input/output parameter which is changed to point to the `Request` whose completion is being reported.

**Return Value** Returns 1 (TRUE) if successful, returns 0 (FALSE) otherwise.

**Notes** CORBA compliant.

**See Also** `CORBA::ORB::send_multiple_requests()`  
`CORBA::Request::get_response()`  
`CORBA::Request::send_deferred()`

### **CORBA::ORB::INFINITE\_TIMEOUT**

**Synopsis** `static const CORBA::ULong CORBA::ORB::INFINITE_TIMEOUT;`

**Description** Used as a parameter to `CORBA::BOA::impl_is_ready()`, `CORBA::BOA::processEvents()`, `CORBA::BOA::processNextEvent()` and `CORBA::BOA::obj_is_ready()` to indicate that a server should not time out waiting for events.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::DEFAULT_TIMEOUT`  
`CORBA::BOA::impl_is_ready()`  
`CORBA::BOA::obj_is_ready()`  
`CORBA::BOA::processEvents()`  
`CORBA::BOA::processNextEvent()`

### **CORBA::ORB::isBaseInterfaceOf()**

**Synopsis** `CORBA::Boolean CORBA::ORB::isBaseInterfaceOf(  
    const char* derived,  
    const char* maybeABase,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description** Determines whether the interface named in `maybeABase` is a base interface of the interface named in `derived`.

**Return Value** Returns 1 (TRUE) if `maybeABase` is a base interface of `derived` (or `maybeABase` and `derived` are the same interface) and returns 0 (FALSE) otherwise.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::baseInterfacesOf()`

---

## CORBA::ORB::isDefaultRxTimeoutSet()

|                     |                                                                                                                                                                                                        |          |                                     |           |                                                                      |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|-------------------------------------|-----------|----------------------------------------------------------------------|
| <b>Synopsis</b>     | <code>CORBA::Boolean CORBA::ORB::isDefaultRxTimeoutSet() const;</code>                                                                                                                                 |          |                                     |           |                                                                      |
| <b>Description</b>  | Returns the current value of the flag to say that the RxTimeout has been set to a value different to the default value.                                                                                |          |                                     |           |                                                                      |
| <b>Return Value</b> | <table> <tr> <td>1 (TRUE)</td> <td>A value for RxTimeout has been set.</td> </tr> <tr> <td>0 (FALSE)</td> <td>The value for RxTimeout has not been set. This is the default value.</td> </tr> </table> | 1 (TRUE) | A value for RxTimeout has been set. | 0 (FALSE) | The value for RxTimeout has not been set. This is the default value. |
| 1 (TRUE)            | A value for RxTimeout has been set.                                                                                                                                                                    |          |                                     |           |                                                                      |
| 0 (FALSE)           | The value for RxTimeout has not been set. This is the default value.                                                                                                                                   |          |                                     |           |                                                                      |
| <b>Exceptions</b>   | If RxTimeout has been set and any fragment of an incoming IOP message is not received within the specified timeout period, a <code>CORBA::COMM_FAILURE</code> exception will be raised.                |          |                                     |           |                                                                      |
| <b>Notes</b>        | Orbix specific.                                                                                                                                                                                        |          |                                     |           |                                                                      |
| <b>See Also</b>     | <code>CORBA::ORB::defaultRxTimeout (CORBA::ULong)</code><br><code>CORBA::ORB::defaultTxTimeout ()</code>                                                                                               |          |                                     |           |                                                                      |

## CORBA::ORB::isForeignFD()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                           |                 |                                                                                        |                   |                                                                                                                                                                                                                                          |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|----------------------------------------------------------------------------------------|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <pre>CORBA::Boolean CORBA::ORB::isForeignFD(     int fd,     unsigned char mask = CORBA::FD_FOREIGN_WRITE                           CORBA::FD_FOREIGN_READ                           CORBA::FD_FOREIGN_EXCEPT);</pre>                                                                                                                                                                                                                     |                 |                                                                                        |                   |                                                                                                                                                                                                                                          |
| <b>Description</b> | Tests if a specified file descriptor has been registered as a foreign file descriptor with the Orbix event processing loop.                                                                                                                                                                                                                                                                                                               |                 |                                                                                        |                   |                                                                                                                                                                                                                                          |
| <b>Parameters</b>  | <table> <tr> <td><code>fd</code></td> <td>The file descriptor to be tested for membership of the registered file descriptor set.</td> </tr> <tr> <td><code>mask</code></td> <td>This parameter allows you to test for which type of events the file descriptor is monitored. The value can be <code>FD_FOREIGN_WRITE</code>, <code>FD_FOREIGN_READ</code>, <code>FD_FOREIGN_EXCEPT</code>, or a combination of these.</td> </tr> </table> | <code>fd</code> | The file descriptor to be tested for membership of the registered file descriptor set. | <code>mask</code> | This parameter allows you to test for which type of events the file descriptor is monitored. The value can be <code>FD_FOREIGN_WRITE</code> , <code>FD_FOREIGN_READ</code> , <code>FD_FOREIGN_EXCEPT</code> , or a combination of these. |
| <code>fd</code>    | The file descriptor to be tested for membership of the registered file descriptor set.                                                                                                                                                                                                                                                                                                                                                    |                 |                                                                                        |                   |                                                                                                                                                                                                                                          |
| <code>mask</code>  | This parameter allows you to test for which type of events the file descriptor is monitored. The value can be <code>FD_FOREIGN_WRITE</code> , <code>FD_FOREIGN_READ</code> , <code>FD_FOREIGN_EXCEPT</code> , or a combination of these.                                                                                                                                                                                                  |                 |                                                                                        |                   |                                                                                                                                                                                                                                          |

**Return Value** Returns 1 (`TRUE`) if the specified file descriptor is part of the foreign file descriptor set *and* is monitored for the specified events. Returns 0 (`FALSE`) otherwise.

**Notes** Orbix specific.

### **CORBA::ORB::isOrbixFD()**

**Synopsis** `CORBA::Boolean CORBA::ORB::isOrbixFD(int fd);`

**Description** Tests if a specified file descriptor is included in the current set of Orbix file descriptors.

#### **Parameters**

`fd` The file descriptor to be tested for membership of the Orbix file descriptor set.

**Return Value** Returns 1 (`TRUE`) if the specified file descriptor is part of the Orbix file descriptor set. Returns 0 (`FALSE`) otherwise.

**Notes** Orbix specific.

### **CORBA::ORB::isOrbixSelectableFD()**

**Synopsis** `CORBA::Boolean CORBA::ORB::isOrbixSelectableFD(int fd);`

**Description** Tests if a specified file descriptor is part of the file descriptor set returned by `CORBA::ORB::getSelectableFDSet()`.

#### **Parameters**

`fd` The file descriptor to be tested for membership of the selectable file descriptor set.

**Return Value** Returns 1 (`TRUE`) if the specified file descriptor is part of the Orbix selectable file descriptor set. Returns 0 (`FALSE`) otherwise.

**Notes** Orbix specific.

---

## CORBA::ORB::list\_initial\_services()

- Synopsis** `CORBA::ObjectIdList_ptr CORBA::ORB::list_initial_services(  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Some services, in particular, CORBA services such as the Naming Service, can only be used by first obtaining an object reference to an object through which the service can be used. (CORBA services are optional extensions to ORB implementations that are specified by CORBA. They include the Naming Service and Event Service.)
- The function `list_initial_services()` finds a list of the services provided by Orbix. Currently only the names “NameService” and “InterfaceRepository” are listed by this function.
- Return Value** Returns a sequence of strings, each of which names a service provided by Orbix.
- Notes** CORBA compliant.
- See Also** `CORBA::ORB::resolve_initial_references()`

## CORBA::ORB::makeIOR()

- Synopsis** `string CORBA::ORB::makeIOR(  
char* host, unsigned short port,  
CORBA::ObjectKey objKey,  
char* typeID,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Creates a string IOR with all the relevant information.
- Parameters** The parameters `host`, `port` and `typeID` are the respective standard elements of the IOR. The `objKey` parameter is the opaque object key part of the IOR required by the CORBA specification.
- The IOR object key is created from the `server`, `interface` and `'orbixhost'` fields of the Orbix object reference, using `CORBA::ORB::makeOrbixObjectKey()`. If the `host` parameter to `makeIOR()` is null, the `'orbixhost'` is also used as the `host` field of the IOR.
- Return Value** Returns an IOR in string form.
- See Also** `CORBA::ORB::makeOrbixObjectKey()`

### **CORBA::ORB::makeOrbixObjectKey()**

- Synopsis** `CORBA::_IDL_SEQUENCE_octet* CORBA::ORB::makeOrbixObjectKey(  
    const char* host,  
    const char* serverName,  
    const char* interfaceName,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Creates an Orbix object key. You can use this to make an IOR.
- Parameters** The object key is created from the server, interface and host fields of the Orbix object reference.
- Return Value** A simple sequence of octets.
- See Also** `CORBA::ORB::makeIOR()`

### **CORBA::ORB::maxConnectRetries()**

- Synopsis** `CORBA::ULong CORBA::ORB::maxConnectRetries(  
    CORBA::ULong retries,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** If an operation call cannot be made on the first attempt because the transport (TCP/IP) connection cannot be established, Orbix retries the attempt every two seconds until either the call can be made or until there have been too many retries. The function `maxConnectRetries()` sets the maximum number of retries. The default number of retries is ten.
- As an alternative, the `Orbix.IT_CONNECT_ATTEMPTS` entry in the Orbix configuration file (`Orbix.cfg`) or as an environment variable may be used to control the maximum number of retries. The value set by `maxConnectRetries()` takes precedence over this. The `Orbix.IT_CONNECT_ATTEMPTS` value is used only if `maxConnectRetries()` is set to 0.
- Return Value** Returns the previous setting.
- Exceptions** A subsequent operation call on an object in the server will raise the system exception `CORBA::COMM_FAILURE` to the client application if the server cannot be contacted within the maximum number of retries.
- Notes** Orbix specific.
- See Also** `CORBA::ORB::connectionTimeout()`

---

## CORBA::ORB::maxConnectRetries()

- Synopsis** `CORBA::ULong CORBA::ORB::maxConnectRetries() const;`
- Description** Returns the current setting of the connectAttempts member variable.
- Return Value** The number of times an operational call attempts a TCP/IP connection to fulfill an operational request. If connectAttempts is equal to 0, then either the configuration file variable Orbix.IT\_CONNECT\_ATTEMPTS or the environment variable IT\_CONNECT\_ATTEMPTS is used. The environment variable takes precedence over the configuration file variable.
- Notes** Orbix specific.
- See Also** `CORBA::ORB::connectionTimeout(CORBA::ULong)`  
`CORBA::ORB::abortSlowConnects(CORBA::Boolean)`

## CORBA::ORB::mustRedefineDeref()

- Synopsis** `CORBA::Boolean CORBA::ORB::mustRedefineDeref(  
CORBA::Boolean onOff,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Sets the mustRedefineDeref flag.
- Parameters**
- |           |                                                                                                                                                                           |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 (TRUE)  | For a given object, the member function <code>CORBA::Object::_deref()</code> must be overridden in a user's interface implementation class, e.g. <code>Account_i</code> . |
| 0 (FALSE) | For a given object, the member function <code>CORBA::Object::_deref()</code> does not have to be overridden. This is the default value.                                   |
- Return Value** Returns the previous setting.
- Notes** Orbix specific.
- See Also** `CORBA::ORB::mustRedefineDeref()`

### **CORBA::ORB::mustRedefineDeref()**

**Synopsis** `CORBA::Boolean CORBA::ORB::mustRedefineDeref() const;`

**Description** Returns the current setting of the `mustRedefineDeref` flag.

**Return Value**

|           |                                                                                                                                                |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 (TRUE)  | The function <code>CORBA::Object::_deref()</code> must be overridden in a user's interface implementation class, e.g. <code>Account_i</code> . |
| 0 (FALSE) | The function <code>CORBA::Object::_deref()</code> does not have to be overridden. This is the default value.                                   |

**Notes** Orbix specific.

**See Also** `CORBA::ORB::mustRedefineDeref(CORBA::Boolean)`

### **CORBA::ORB::myHost()**

**Synopsis** `const char* CORBA::ORB::myHost(  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())  
const;`

**Description** Returns the name of the host on which the `CORBA::Orbix` object is located.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::myServer()`

### **CORBA::ORB::myServer()**

**Synopsis** `const char* CORBA::ORB::myServer(  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())  
const;`

**Description** Returns the server name for which the `CORBA::Orbix` object was created. If called from a client it returns the process identifier in string form.

**Notes** Orbix specific.



---

**See Also**      `CORBA::ORB::myHost()`  
                  `CORBA::ORB::setServerName()`

## **CORBA::ORB::noReconnectOnFailure()**

**Synopsis**      `CORBA::Boolean CORBA::ORB::noReconnectOnFailure(  
                  CORBA::Boolean dont_reconnect,  
                  CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description**      When an Orbix client first contacts a server, a single communications channel is established between the client-server pair. This connection is then used for all subsequent communications between the client and the server. The connection is closed only when either the client or the server exits.

When a server exits while a client is still connected, the next invocation by the client raises a system exception of type `CORBA::COMM_FAILURE`. If the client attempts another invocation, Orbix automatically tries to re-establish the connection.

---

**Note:** Orbix automatically tries to re-establish without throwing a system exception.

---

You can change this default behavior by passing the value 1 (`TRUE`) to the function `CORBA::ORB::noReconnectOnFailure()`. Then, all client attempts to contact a server subsequent to closure of the communications channel raise a `CORBA::COMM_FAILURE` system exception.

### **Parameters**

- |                          |                                                                                                                                                        |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 ( <code>TRUE</code> )  | All client attempts to contact a server subsequent to closure of the communications channel raise a <code>CORBA::COMM_FAILURE</code> system exception. |
| 0 ( <code>FALSE</code> ) | Orbix attempts to re-establish a failed connection to a server on the next invocation by a client. This is the default behavior.                       |

**Return Value**      Returns the previous value.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::maxConnectRetries()`

### **CORBA::ORB::noReconnectOnFailure()**

**Synopsis** `CORBA::Boolean CORBA::ORB::noReconnectOnFailure() const;`

**Description** Returns the current value of the `noReconnectOnFailure` flag.

**Return Value**

|           |                                                                                                                                                            |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 (TRUE)  | A <code>CORBA::COMM_FAILURE</code> system exception will be raised if a client has lost its connection with a server and then tries to contact the server. |
| 0 (FALSE) | Orbix attempts to re-establish a failed connection to a server on the next invocation by a client. This is the default value.                              |

**Notes** Orbix specific.

**See Also** `CORBA::ORB::noReconnectionOnFailure(CORBA::Boolean)`

### **CORBA::ORB::object\_to\_string()**

**Synopsis**

```
char* CORBA::ORB::object_to_string(
 CORBA::Object_ptr obj,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Converts an object reference to a string.

A stringified Orbix object reference is of the form:

```
: \host:serverName:marker:IFR_host:IFR_Server
:interfaceMarker
```

These fields are as follows:

|      |                                     |
|------|-------------------------------------|
| host | The host name of the target object. |
|------|-------------------------------------|

---

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>serverName</code>       | The name of the target object's server as registered in the Implementation Repository and also as specified to <code>CORBA::BOA::impl_is_ready()</code> , <code>CORBA::BOA::object_is_ready()</code> or set by <code>setServerName()</code> . For a local object in a server, this is that server's name (if that server's name is known), otherwise it is the process' identifier. The server name is known if the server is launched by Orbix; or if it is launched manually and the server name is passed to <code>impl_is_ready()</code> or if the server name has been set by <code>CORBA::ORB::setServerName()</code> . |
| <code>marker</code>           | The object's marker name. This can be chosen by the application, or it is a string of digits chosen by Orbix.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <code>IFR_host</code>         | The name of a host running an Interface Repository that stores the target object's IDL definition. Normally, this is blank.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <code>IFR_server</code>       | The string "IFR".                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>interface-Marker</code> | The target object's interface. If called on a proxy, this may not be the object's true (most derived) interface—it may be a base interface.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

**Return Value** Returns an Orbix stringified object reference. The caller is responsible for deleting the string returned.

**Notes** CORBA compliant.

**See Also** `CORBA::ORB::string_to_object()`  
`CORBA::Object::_object_to_string()`

## **CORBA::ORB::optimiseProtocolEncoding()**

**Synopsis**

```
CORBA::Boolean CORBA::ORB::optimiseProtocolEncoding(
 CORBA::Boolean value,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Enables protocol encoding optimisations if `value` is set to `TRUE`, which is the default value. Protocol encoding optimisations are disabled if `value` set to `FALSE`. Protocol encoding optimisations are implemented within Orbix by the use of null encoding (that is, no encoding at all) when appropriate, to transmit data.

**Return Value** Returns the previous setting.

**Notes** Orbix specific. You are unlikely to need to use this function.

### **CORBA::ORB::optimiseProtocolEncoding()**

**Synopsis** `CORBA::Boolean CORBA::ORB::optimiseProtocolEncoding() const;`

**Description** Returns the current setting of the `optimiseProtocolEncoding` flag.

#### **Return Value**

|           |                                                                         |
|-----------|-------------------------------------------------------------------------|
| 1 (TRUE)  | When <code>optimiseProtocolEncoding</code> (null encoding) is used.     |
| 0 (FALSE) | When <code>optimiseProtocolEncoding</code> (null encoding) is not used. |

**Notes** Orbix specific. You are unlikely to need this function.

**See Also** `CORBA::ORB::optimiseProtocolEncoding(CORBA::Boolean)`.

### **CORBA::ORB::Output()**

#### **Synopsis**

```
static void CORBA::ORB::Output(
 const char* string, int level = 1);
static void CORBA::ORB::Output(
 CORBA::Environment& e, int level = 1);
static void CORBA::ORB::Output(
 CORBA::SystemException* x, int level = 1);
```

#### **Description**

A set of functions to output application's diagnostic and other output via the active output handlers. Unless overridden by an implementation of class `CORBA::ORB::UserOutput`, all output is directed to the C++ `cout` stream via the default output handler, `IT_StdOutHandler`.

#### **Parameters**

|                      |                                                               |
|----------------------|---------------------------------------------------------------|
| <code>message</code> | Outputs a string.                                             |
| <code>env</code>     | Outputs details of an Environment.                            |
| <code>sysEx</code>   | Outputs details of a system exception.                        |
| <code>level</code>   | The diagnostic level. (All Orbix output is assigned level 1.) |

---

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <b>Notes</b>    | Orbix specific.                                 |
| <b>See Also</b> | CORBA::UserOutput<br>CORBA::ORB::setDiagnostics |

### CORBA::ORB::pingDuringBind()

|                 |                                                                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b> | <pre>CORBA::ORB::Boolean pingDuringBind(     CORBA::Boolean pingOn,     CORBA::Environment&amp; IT_env = CORBA::IT_chooseDefaultEnv());</pre> |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------|

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <p>By default, <code>_bind()</code> raises an exception if the object on which the <code>_bind()</code> is attempted is unknown to Orbix. Doing so requires Orbix to ping the desired object (the ping operation is defined by Orbix and it has no effect on the target object). The pinging causes the target server process to be activated if necessary, and confirms that this server recognizes the target object. You can disable pinging using <code>pingDuringBind()</code>, by passing <code>FALSE</code> to the parameter <code>pingOn</code>.</p> |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

You may wish to disable pinging to improve efficiency by reducing the overall number of remote invocations.

When `pingDuringBind(FALSE)` is called:

- ◆ A `_bind()` to an unavailable object does not immediately raise an exception, but subsequent requests using the object reference returned from `_bind()` will fail by raising a system exception (`CORBA::INV_OBJREF`).
- ◆ The `_bind()` does not itself make any remote calls; it simply sets up a proxy with the required fields.

|                     |                               |
|---------------------|-------------------------------|
| <b>Return Value</b> | Returns the previous setting. |
|---------------------|-------------------------------|

|              |                 |
|--------------|-----------------|
| <b>Notes</b> | Orbix specific. |
|--------------|-----------------|

### CORBA::ORB::pingDuringBind()

|                 |                                                               |
|-----------------|---------------------------------------------------------------|
| <b>Synopsis</b> | <pre>CORBA::Boolean CORBA::ORB::pingDuringBind() const;</pre> |
|-----------------|---------------------------------------------------------------|

|                    |                                                                      |
|--------------------|----------------------------------------------------------------------|
| <b>Description</b> | Returns the current setting of the <code>pingDuringBind</code> flag. |
|--------------------|----------------------------------------------------------------------|

### Return Values

|           |                                                        |
|-----------|--------------------------------------------------------|
| 1 (TRUE)  | If pings are used when attempting to bind to a server. |
| 0 (FALSE) | If pings are not used.                                 |

**Notes** Orbix specific.

**See Also** `CORBA::ORB::pingDuringBind(CORBA::Boolean)`

### **CORBA::ORB::PlaceCVHandlerAfter()**

#### Synopsis

```
static void CORBA::ORB::PlaceCVHandlerAfter(
 const char* handler,
 const char* afterThisHandler);
```

#### Description

Modifies the order in which configuration handlers are called. If not explicitly rearranged, configuration value handlers are called in reverse order of the order in which they are instantiated in an application.

You must call the function `ReinitialiseConfig()` before modifications by this function take effect.

#### Parameters

|                               |                                                                                                                           |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>handler</code>          | The handler to be rearranged.                                                                                             |
| <code>afterThisHandler</code> | The configuration value handler after which the handler specified in the parameter <code>handler</code> should be placed. |

**Notes** Orbix specific.

**See Also** `CORBA::ORB::PlaceCVHandlerBefore()`  
`CORBA::ORB::ReinitialiseConfig()`  
`CORBA::UserCVHandler`  
`CORBA::ExtraRegistryCVHandler`  
`CORBA::ExtraConfigFileHandler`

---

## CORBA::ORB::PlaceCVHandlerBefore()

**Synopsis**

```
static void CORBA::ORB::PlaceCVHandlerBefore(
 const char* handler,
 const char* beforeThisHandler);
```

**Description**

Modifies the order in which configuration handlers are called. If not explicitly rearranged, configuration value handlers are called in reverse order of the order in which they are instantiated in an application.

The function `ReinitialiseConfig()` must be called before modifications by this function take effect.

**Parameters**

|                                |                                                                                                                            |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <code>handler</code>           | The handler to be rearranged.                                                                                              |
| <code>beforeThisHandler</code> | The configuration value handler before which the handler specified in the parameter <code>handler</code> should be placed. |

**Notes**

Orbix specific.

**See Also**

`CORBA::ORB::PlaceCVHandlerAfter()`  
`CORBA::ORB::ReinitialiseConfig()`  
`CORBA::UserCVHandler`  
`CORBA::ExtraRegistryCVHandler`  
`CORBA::ExtraConfigFileHandler`

## CORBA::ORB::poll\_next\_response()

**Synopsis**

```
CORBA::Boolean CORBA::ORB::poll_next_response(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description**

May be called successively to determine whether the individual requests specified in a `CORBA::ORB::send_multiple_requests_oneway()` or `CORBA::ORB::send_multiple_requests_deferred()` call have completed successfully. The function returns immediately.

Alternatively you may call the function `CORBA::Request::poll_response()` on the individual Request objects in the sequence of Requests passed to `send_multiple_requests_oneway()` and `send_multiple_requests_deferred()`.

**Notes** CORBA compliant.

**See Also** `CORBA::ORB::get_next_response()`  
`CORBA::ORB::send_multiple_requests_oneway()`  
`CORBA::Request::poll_response()`

### **CORBA::ORB::registerIOCallback()**

**Synopsis** `CORBA::OrbixIOCallback CORBA::ORB::registerIOCallback(  
CORBA::OrbixIOCallback IOCallback,  
CORBA::OrbixIOCallbackType IOCallbackType);`

**Description** An application may wish to be informed when a new connection is established, or when an existing connection is closed. A connection is opened when a client first communicates with the server; and it is closed when the client terminates or the communication's level reports a break in service between the server and client. A client or server application may specify functions to be called at either of these two events by calling the function `registerIOCallback()` on the `CORBA::Orbix` object. You can disable callbacks by passing 0 as the first parameter.

#### **Parameters**

`IOCallback` The type of this parameter is specified by the type definition:

```
// C++
typedef void
(*OrbixIOCallback) (int);
```

`IOCallback` is a pointer to a function that takes an `int` parameter and returns `void`. This function will be called when either an open connection or a close connection event occurs, depending on the value of the second parameter, `IOCallbackType`. The parameter passed to the automatically-called function indicates the file descriptor assigned to the connection that is either being opened or closed. While handling an operation invocation, a server can use the function `CORBA::Request::descriptor()` to find the file descriptor assigned to the then current connection:



```
// C++
CORBA::Request* my_request =
 CORBA::Environment::request();
my_request->descriptor();
```

The `int` value returned from this call is subsequently passed to the function automatically called by `registerIOCallback()` when the connection to that client closes.

`IOCallbackType` The type of event for which the application would like to receive callbacks. The parameter `IOCallbackType` takes one of the values, `FD_OPEN_CALLBACK` or `FD_CLOSE_CALLBACK`, defined in the enumerated type `IOCallbackType`.

**Return Value** Returns the address of the previous function that was to be called when a connection was opened or closed.

**Notes** Orbix specific.

**See Also** `CORBA::Request::descriptor()`  
`CORBA::BOA::getFileDescriptors()`

## **CORBA::ORB::registerIOCallbackObject()**

**Synopsis** `CORBA::Boolean CORBA::ORB::registerIOCallbackObject(
 unsigned char eventType, IT_IOCallback* obj);`

**Description** As described in the reference for class `CORBA::IT_IOCallback`, Orbix allows you to receive callbacks when Orbix connections are opened or closed and when events occur on foreign file descriptors added to the Orbix event loop. To receive these callbacks, you must implement a class that inherits from `CORBA::IT_IOCallback` and register an object of this type with Orbix. The function `CORBA::ORB::registerIOCallback()` allows you to register a callback object with Orbix.

### Parameters

|                        |                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>eventType</code> | Orbix can monitor both native Orbix file descriptors and foreign file descriptors for events, as described in <code>CORBA::ORB::addForeignFd()</code> . This parameter indicates which file descriptors should be monitored with respect to the callback object being registered. The value of this parameter is <code>CORBA::OrbixIO</code> , <code>CORBA::ForeignIO</code> , or a logical combination of these. |
| <code>obj</code>       | The callback object to be registered with Orbix.                                                                                                                                                                                                                                                                                                                                                                  |

### Return Value

|           |                                           |
|-----------|-------------------------------------------|
| 1 (TRUE)  | If the object is successfully registered. |
| 0 (FALSE) | Otherwise.                                |

**Notes** Orbix specific.

**See Also** `CORBA::ORB::addForeignFD()`  
`CORBA::ORB::unregisterIOCallback()`  
`CORBA::IT_IOCallback`

## **CORBA::ORB::registerPerObjectServiceContextHandler()**

**Synopsis**

```
CORBA::Boolean CORBA::ORB::registerPerObjectServiceContextHandler(
 CORBA::ServiceContextHandler* CtxHandler,
 CORBA::Object* theObject,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Registers a per-object service context handler with Orbix by adding it to the list of Service Context Handlers internally stored.

### Parameters

|                         |                                         |
|-------------------------|-----------------------------------------|
| <code>CtxHandler</code> | The handler to be registered.           |
| <code>theObject</code>  | The object to register the handler for. |

**Return Value** 1 (TRUE) if successful; 0 (FALSE) otherwise.

**See Also** `ServiceContextHandler` class.

## **CORBA::ORB::registerPerRequestServiceContextHandler()**

**Synopsis** `CORBA::Boolean`  
`CORBA::ORB::registerPerRequestServiceContextHandler(`  
`CORBA::ServiceContextHandler* CtxHandler,`  
`CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description** Registers a per-request service context handler with Orbix by adding it to the list of Service Context Handlers internally stored.

### **Parameters**

`CtxHandler` The handler to be registered.

**Return Value** 1 (TRUE) if successful; 0 (FALSE) otherwise.

**See Also** `ServiceContextHandler` class.

## **CORBA::ORB::ReinitialiseConfig()**

**Synopsis** `static void CORBA::ORB::ReinitialiseConfig();`

**Description** This function must be called to make effective any modifications to the arrangement or activation of configuration value handlers. In particular, it must be called in order that changes made by `ActivateCVHandler()`, `DeactivateCVHandler()`, `PlaceCVHandlerBefore()`, `PlaceCVHandlerAfter()` take effect.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::PlaceCVHandlerBefore()`  
`CORBA::ORB::PlaceCVHandlerAfter()`  
`CORBA::ORB::ActivateCVHandler()`  
`CORBA::ORB::DeactivateCVHandler()`  
`CORBA::ORB::SetConfigValue()`  
`CORBA::UserCVHandler`  
`CORBA::ExtraRegistryCVHandler`  
`CORBA::ExtraConfigFileHandler`

### **CORBA::ORB::removeForeignFD()**

**Synopsis**

```
void CORBA::ORB::removeForeignFD(
 const int fd, unsigned char aState);
```

**Description**

If you have added foreign file descriptors to the Orbix event loop using `CORBA::ORB::addForeignFD()` or `CORBA::ORB::addForeignFDSet()`, this function allows you to remove a single foreign file descriptor.

**Parameters**

`fd`            The file descriptor to be removed from the Orbix event processing loop.

`aState`        Indicates the type of events for which the file descriptor should no longer be monitored by Orbix. This can be `FD_READ`, `FD_WRITE`, `FD_EXCEPT`, or a logical combination of these values.

**Notes**

Orbix specific.

**See Also**

```
CORBA::ORB::addForeignFD()
CORBA::ORB::addForeignFDSet()
CORBA::ORB::removeForeignFDSet()
```

### **CORBA::ORB::removeForeignFDSet()**

**Synopsis**

```
void CORBA::ORB::removeForeignFDSet(
 fd_set& theFDset, unsigned char aState);
```

**Description**

If you have added foreign file descriptors to the Orbix event loop using `CORBA::ORB::addForeignFD()` or `CORBA::ORB::addForeignFDSet()`, this function allows you to remove a set of foreign file descriptors.

**Parameters**

`fds`            The file descriptor set to be removed from the Orbix event processing loop.

`aState`        Indicates the type of events for which the file descriptors should no longer be monitored by Orbix. This can be `FD_READ`, `FD_WRITE`, `FD_EXCEPT`, or a logical combination of these values.

**Notes**

Orbix specific.

**See Also** CORBA::ORB::addForeignFD()  
 CORBA::ORB::addForeignFDSet()  
 CORBA::ORB::removeForeignFD()

## CORBA::ORB::reSizeObjectTable()

**Synopsis** void CORBA::ORB::reSizeObjectTable(  
     CORBA::ULong newSize,  
     CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());

**Description** Resizes the object table. All of the Orbix objects in an address space are registered in its object table—a hash table that maps from object identifiers to the location of objects in virtual memory. It may be important that this table is not too small for the number of objects in the process, since long overflow chains lead to inefficiencies. The default size of the object table is defined as the value:

CORBA::\_OBJECT\_TABLE\_SIZE\_DEFAULT

in the file CORBA.h.

If you anticipate that a program will handle a much larger number of objects than the default size (which is of the order of 1000), you can use this function to resize the table.

### Parameters

`newSize` The value given to `newSize` should be in the same order as the number of objects expected to be managed in the process.

**Notes** Orbix specific.

**See Also** CORBA::\_OBJECT\_TABLE\_SIZE\_DEFAULT

## CORBA::ORB::resolve\_initial\_references()

**Synopsis** CORBA::Object\_ptr CORBA::ORB::resolve\_initial\_references(  
     const char\* identifier,  
     CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());

**Description** Returns an object reference through which a service (for example, Interface Repository or a CORBA service such as the Naming Service) can be used.

### Parameters

`identifier` The name of the desired service. A list of services supported by Orbix can be obtained using `CORBA::ORB::list_initial_services()`.

**Return Value** Returns an object reference for the desired service. The object reference returned must be narrowed to the correct object type. For example, the object reference returned from resolving the name “InterfaceRepository” must be narrowed to the type `CORBA::Repository`.

**Notes** CORBA compliant.

**See Also** `CORBA::ORB::list_initial_services()`

### **CORBA::ORB::resortToStatic()**

#### Synopsis

```
CORBA::Boolean resortToStatic(CORBA::Boolean value,
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

#### Description

When a reference to a remote object enters a client's or server's address space, Orbix constructs a proxy for that object. This proxy (a normal C++ object) is constructed to execute the proxy code corresponding to the actual interface of the true object it represents. Hence if a server object has an operation of the form:

```
// IDL
// In some interface.
void op(in Account a);
```

and if a reference to a (remote) `CurrentAccount` (a derived interface of `Account`) is passed as a parameter to this operation, Orbix tries to set up a proxy for a `CurrentAccount` in the server's address space.

If the server was not linked with the IDL compiler generated proxy code for `CurrentAccount`, Orbix creates a proxy for an `Account` in the server's address space. That is, Orbix uses the static rather than the dynamic type of the parameter. The same applies when an object reference enters a client.

If resorting to the static type is not acceptable, you should call `resortToStatic()` on the `CORBA::Orbix` object, passing `0` (`FALSE`) for the first parameter. The default setting is `1` (`TRUE`). Setting the value to `0` (`FALSE`) means that Orbix raises an exception if the server or client is not linked with the actual (dynamic) proxy code.

|                     |                                              |
|---------------------|----------------------------------------------|
| <b>Return Value</b> | Returns the previous setting.                |
| <b>Notes</b>        | Orbix specific.                              |
| <b>See Also</b>     | <code>CORBA::ORB::useRemoteIsACalls()</code> |

### **CORBA::ORB::resortToStatic()**

|                    |                                                                                                                                                         |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <code>CORBA::Boolean CORBA::ORB::resortToStatic() const;</code>                                                                                         |
| <b>Description</b> | Returns the current setting of the <code>resortToStatic</code> flag. See " <code>CORBA::ORB::resortToStatic()</code> " for an explanation of this flag. |

#### **Return Values**

|                                       |                                                                                                                                                                                           |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>1</code> ( <code>TRUE</code> )  | When <code>resortToStatic</code> is used for unknown derived interfaces and the base interface is known by the CORBA process. This is the default value.                                  |
| <code>0</code> ( <code>FALSE</code> ) | When <code>resortToStatic</code> is not being used and Orbix raises an exception if the CORBA process has not been linked with the IDL compiler generated code for the derived interface. |

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <b>Notes</b>    | Orbix specific.                                         |
| <b>See Also</b> | <code>CORBA::ORB::resortToStatic(CORBA::Boolean)</code> |

### **CORBA::ORB::send\_multiple\_requests\_deferred()**

|                 |                                                                                                                                                                                        |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b> | <code>CORBA::Status CORBA::ORB::send_multiple_requests_deferred(<br/>const CORBA::RequestSeq&amp; requests,<br/>CORBA::Environment&amp; IT_env = CORBA::IT_chooseDefaultEnv());</code> |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Description** Initiates a number of requests in parallel. It does not wait for the requests to finish before returning to the caller. The caller can use `CORBA::get_next_response()` or `CORBA::Request::get_response()` to determine the outcome of the requests. Memory leakage results if one of these functions is not called for a request issued with `CORBA::Request::send()` or `CORBA::ORB::send_multiple_requests()`.

**Parameters**

`requests` A sequence of requests.

**Return Value** 1 (TRUE) if successful; 0 (FALSE) otherwise.

**Notes** CORBA compliant.

**See Also** `CORBA::ORB::send_multiple_requests_oneway()`  
`CORBA::Request::get_response()`  
`CORBA::Request::send_deferred()`  
`CORBA::ORB::send_multiple_requests_oneway()`  
`CORBA::ORB::poll_next_response()`

### **CORBA::ORB::send\_multiple\_requests\_oneway()**

**Synopsis**

```
CORBA::Status CORBA::ORB::send_multiple_requests_oneway(
 const CORBA::RequestSeq& requests,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Initiates a number of requests in parallel. It does not wait for the requests to finish before returning to the caller.

**Parameters**

`requests` A sequence of requests. The operations in this sequence do not have to be IDL oneway operations. The caller does not expect a response, nor does it expect `out` or `inout` parameters to be updated.

**Return Value** 1 (TRUE) if successful; 0 (FALSE) otherwise.

**Notes** CORBA compliant.

**See Also** `CORBA::Request::send_oneway()`  
`CORBA::ORB::send_multiple_requests_deferred()`



---

## CORBA::ORB::SetConfigValue()

**Synopsis**

```
static CORBA::Boolean CORBA::ORB::SetConfigValue(
 const char* name, char* value);
```

**Description**

Sets the specified configuration entry in this process. (It does not set the configuration entry in the Orbix configuration file or system registry.)

The configuration entries that can be changed by `SetConfigValue()` are:

- ◆ `Common.IT_IMP_REP_PATH`
- ◆ `Common.IT_INT_REP_PATH`
- ◆ `Orbix.ENABLE_ANON_BIND_SUPPORT`
- ◆ `Orbix.IT_ACT_POLICY`
- ◆ `Orbix.IT_CONNECT_ATTEMPTS`
- ◆ `Orbix.IT_DEF_NUM_NW_THREADS`
- ◆ `Orbix.IT_ERRORS`
- ◆ `Orbix.IT_FD_STOP_LISTENING_POINT`
- ◆ `Orbix.IT_FD_WARNING_NUMBER`
- ◆ `Orbix.IT_LISTEN_QUEUE_SIZE`
- ◆ `Orbix.IT_MARKER_PATTERN`
- ◆ `Orbix.IT_ONEWAY_RESPONSE_REQUIRED`
- ◆ `Orbix.IT_SERVER_CODE`
- ◆ `Orbix.IT_SERVER_COMMS`
- ◆ `Orbix.IT_SERVER_MARKER`
- ◆ `Orbix.IT_SERVER_METHOD`
- ◆ `Orbix.IT_SERVER_METHOD`
- ◆ `Orbix.IT_SERVER_NAME`
- ◆ `Orbix.IT_SERVER_PORT`
- ◆ `Orbix.IT_USE_ORBIX3_STYLE_SYS_EXC`
- ◆ `Orbix.IT_USE_REVERSE_LOOKUP`
- ◆ `OrbixNames.IT_NAMES_SERVER_HOST`

**Parameters**

`name` The name of the configuration item.

`value` The value to be assigned to the configuration item. Orbix makes a copy of the value passed in.

**Return Value** 1 (TRUE) if successful; 0 (FALSE) otherwise.

**Notes** Orbix specific.

### **CORBA::ORB::SetConfigValueBool()**

**Synopsis**

```
static CORBA::Boolean CORBA::ORB::SetConfigValueBool(
 const char* name, CORBA::Boolean value);
```

**Description** Sets the specified configuration entry in this process. (It does not set the configuration entry in the Orbix configuration file or system registry.)

The configuration entries that can be changed by `SetConfigValue()` are listed with "`CORBA::ORB::SetConfigValue()`" on page 249.

#### **Parameters**

`name` The name of the configuration item.

`value` The value to be assigned to the configuration item.

**Return Value** 1 (TRUE) if successful; 0 (FALSE) otherwise.

**Notes** Orbix specific.

### **CORBA::ORB::SetConfigValueLong()**

**Synopsis**

```
static CORBA::Boolean CORBA::ORB::SetConfigValueLong(
 const char* name, CORBA::Long value);
```

**Description** Sets the specified configuration entry in this process. (It does not set the configuration entry in the Orbix configuration file or system registry.)

The configuration entries that can be changed by `SetConfigValue()` are listed with "`CORBA::ORB::SetConfigValue()`" on page 249.

#### **Parameters**

`name` The name of the configuration item.

`value` The value to be assigned to the configuration item.

**Return Value** 1 (TRUE) if successful; 0 (FALSE) otherwise.

**Notes** Orbix specific.

### **CORBA::ORB::setDiagnostics()**

**Synopsis** `CORBA::Short CORBA::ORB::setDiagnostics(CORBA::Short level,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description** Controls the level of diagnostic messages output to the `cout` stream by the Orbix libraries. The previous setting is returned.

| <b>Level</b> | <b>Output</b>                                  |
|--------------|------------------------------------------------|
| 0            | No diagnostics                                 |
| 1            | Simple diagnostics. This is the default value. |
| 2            | Full diagnostics                               |
| 3            | Full diagnostics with additional IIOP tracing. |

You can obtain an interleaved history of activity across the distributed system from the full diagnostic output, say, from a client to a server, by redirecting the diagnostic messages from both the client and the server to files and then sorting a merged copy of these files.

**Return Value** Returns the previous setting.

**Notes** Orbix specific.

### **CORBA::ORB::setMyReqTransformer()**

**Synopsis** `CORBA::IT_reqTransformer*  
CORBA::IT_reqTransformer::setMyReqTransformer(  
CORBA::IT_reqTransformer*,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description** Registers an `IT_reqTransformer` object as the default transformation for Requests leaving or entering the address space.

#### **Parameters**

`transformer` A pointer to the transformer object.

**Return Value** Returns a pointer to the previous transformer registered for this process; returns zero if no transformer previously registered.

**Notes** Orbix specific.

**See Also** CORBA::IT\_reqTransformer  
CORBA::setReqTransformer()

### CORBA::ORB::setReqTransformer()

**Synopsis**

```
void CORBA::IT_reqTransformer::setReqTransformer(
 CORBA::IT_reqTransformer* transformer,
 const char* server,
 const char* host = 0,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Registers a transformer to be used when sending or receiving data from a specific server on a specific host. A transformer registered using this function overrides any transformer registered for the process when communicating with the server and host specified.

#### Parameters

|             |                                                                 |
|-------------|-----------------------------------------------------------------|
| transformer | A pointer to the transformer to be registered.                  |
| server      | The name of the server for which the transformer is to be used. |
| host        | The name of the host.                                           |

**Notes** Orbix specific.

**See Also** CORBA::setMyReqTransformer()

### CORBA::ORB::setServerName()

**Synopsis**

```
void CORBA::ORB::setServerName(
 const char* serverName,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Sets the server name for the current server process. This function may be used to overcome the problem of exporting object references from a persistent server before calling `impl_is_ready()`.

---

**Note:** If you are using callbacks, you should not invoke `setServerName()` from a client. If a client invokes `setServerName()`, server operations on its callback object fails.

---

**Notes** Orbix specific.

**See Also** `CORBA::ORB::myServer()`  
`CORBA::ORB::impl_is_ready()`  
`CORBA::BOA::change_implementation()`

### **CORBA::ORB::set\_unsafeDelete()**

**Synopsis** `CORBA::IT_PFV CORBA::ORB::set_unsafeDelete(IT_PFV pfv);`

**Description** If you call the C++ `delete` operator (implicitly or explicitly) on an Orbix object with a reference count not equal to one, Orbix issues an error message and immediately exits. You can specify an alternative action using `set_unsafeDelete()`. The function specified in the parameter `pfv` is called if `delete` is called on an Orbix object with a reference count not equal to one.

**Parameters**

`pfv` A pointer to a function that takes no parameters and has a void return type.

**Return Value** Returns the address of the previous function that was to be called on a delete error.

**Notes** Orbix specific.

### **CORBA::ORB::set\_unsafeFDClose()**

**Synopsis** `typedef unsigned char(*IT_PFB) (fd_set&);`  
`IT_PFB CORBA::ORB::set_unsafeFDClose(IT_PFB pfb);`

**Description** No file descriptor (FD) that Orbix manages should be closed with Orbix being unaware of the closure. However, since the user has access to the Orbix FDs it is possible that the user can directly close an FD without using an Orbix API. This is illegal, and will be detected by an unrecoverable failure of the low-level TCP/IP `select()` call. If such a failure occurs, Orbix nominally issues an error message

and immediately exits. If there is a problem with a file descriptor, Orbix issues an error message and immediately exits. The user is able to specify alternative action using `set_unsafeFDclose()`. The user defined function specified in the parameter `pfb` is called if a problem is found with any of the file descriptors being used. This applies to all FD's, not just Foreign FD's

### Parameters

`pfb` A pointer to a function that takes a reference to an `fd_set` as a parameter and returns a pointer to an unsigned char.

The user defined function should have a signature as per the following code fragment and return a pointer to an unsigned char:

```
unsigned char cleanUpFDSet(fd_set& my_fd_set)
{
 // do something to my_fd_set
 if (now_happy_with_fd_set)
 {
 return 1;
 }
}
```

The unsigned char should contain either:

- 1 (TRUE) if the modifications to the `fd_set` are successful; or
- 0 (FALSE) if the `fd_set` modifications were unsuccessful.

**Return Value** A pointer to the last previously supplied function, if any. Otherwise the null pointer.

**Notes** Orbix specific. This function may not supported in future releases.

**See Also** `CORBA::ORB::getAllOrbixFDs()`

### **CORBA::ORB::string\_to\_object()**

**Synopsis** `CORBA::Object_ptr CORBA::ORB::string_to_object(
 const char* ior,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`

**Description** Converts the stringified object reference `obj_ref_string` to an object reference. A stringified object reference is of the form:

```
:\host:serverName:marker:IFR_host:IFR_server
:interfaceMarker
```

Refer to `CORBA::ORB::object_to_string()` for a description of these fields. The target object may not exist (it is not pinged).

**Return Value** Returns an object reference. This may be a null object if the string passed in the parameter `obj_ref_string` is not a recognized stringified object reference format.

**Notes** CORBA compliant.

**See Also**

```
CORBA::ORB::string_to_object(
 const char* host,
 const char* impl,
 const char* marker,
 const char* intfHost,
 const char* intfImpl,
 const char* intfMarker),
CORBA::ORB::object_to_string()
CORBA::ORB::pingDuringBind()
CORBA::Object::Object(const char* obj_ref_string);
```

## CORBA::ORB::string\_to\_object()

**Synopsis**

```
CORBA::Object_ptr CORBA::ORB::string_to_object(
 const char* host,
 const char* impl,
 const char* marker,
 const char* intfHost,
 const char* intfImpl,
 const char* intfMarker,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Creates an object from the strings given as arguments to an object reference.

### Parameters

|                     |                                         |
|---------------------|-----------------------------------------|
| <code>host</code>   | The host name of the target object.     |
| <code>impl</code>   | The name of the target object's server. |
| <code>marker</code> | The object's marker name.               |

|                         |                                                                                                    |
|-------------------------|----------------------------------------------------------------------------------------------------|
| <code>intfHost</code>   | The name of a host running an Interface Repository that stores the target object's IDL definition. |
| <code>intfImpl</code>   | The string "IFR".                                                                                  |
| <code>intfMarker</code> | The target object's interface.                                                                     |

**Return Value** Returns an Orbix object reference constructed from the parameters passed to the function.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::string_to_object()`  
`CORBA::ORB::object_to_string()`  
`CORBA::Object::Object()`

### **CORBA::ORB::supportBidirectionalIIOP()**

**Synopsis**

```
CORBA::Boolean CORBA::ORB::supportBidirectionalIIOP(
 CORBA::Boolean on,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** When an Orbix client connects to an Orbix server, Orbix opens a single connection through which all communications from the client to the server pass. If the server then obtains a reference to an object located in the client, *and* the client and server communicate using the CORBA Internet Inter-ORB Protocol (IIOP), Orbix opens a second connection from the server to the client. If the server attempts to call operations on the client object, this second connection is used.

In some circumstances, for example when using PROGRESS Software's Orbix Wonderwall product, it may be useful to allow all IIOP communications to travel in both directions between client and server across a *single* connection. This function allows you to specify whether or not Orbix should use this form of bidirectional IIOP communications.

#### **Parameters**

`on` 1 (TRUE) enables bidirectional IIOP. 0 (FALSE) disables bidirectional IIOP. By default, bidirectional IIOP is disabled.

**Return Value** Returns the previous setting.



**Notes** Orbix specific.

### **CORBA::ORB::supportBidirectionalIIOP()**

**Synopsis** `CORBA::Boolean CORBA::ORB::supportBidirectionalIIOP() const;`

**Description** Returns the current setting of the supportBidirectionalIIOP flag.

**Return Value**

|           |                                                                                                                                                 |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 (TRUE)  | If the Orbix server obtains a reference to an object located in the client, the server attempts to reuse the existing connection to the client. |
| 0 (FALSE) | The Orbix server opens a new connection when contacting the client. This is the default value.                                                  |

**Notes** Orbix specific.

**See Also** `CORBA::ORB::supportBidirectionalIIOP(CORBA::Boolean)`

### **CORBA::ORB::unregisterIOCallbackObject()**

**Synopsis** `CORBA::Boolean CORBA::ORB::unregisterIOCallbackObject(  
    unsigned char eventType,  
    CORBA::IT_IOCallback* obj);`

**Description** This function removes a callback object registered using `CORBA::ORB::registerIOCallbackObject()`.

**Parameters**

|                        |                                                                                                                                                                                                                                                                       |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>eventType</code> | This parameter indicates which file descriptors should no longer be monitored with respect to the callback object being unregistered. The value of this parameter is <code>CORBA::OrbixIO</code> , <code>CORBA::ForeignIO</code> , or a logical combination of these. |
| <code>obj</code>       | The callback object to be unregistered.                                                                                                                                                                                                                               |

**Return Value** Returns 1 (TRUE) value if the object is successfully unregistered. Returns 0 (FALSE) otherwise.

**Notes** Orbix specific.

**See Also** CORBA::ORB::addForeignFD()  
CORBA::ORB::registerIOCallback()  
CORBA::IT\_IOCallback

### **CORBA::ORB::unregisterPerObjectServiceContext Handler()**

**Synopsis** CORBA::Boolean  
CORBA::ORB::unregisterPerObjectServiceContextHandler(  
CORBA::ULong& CtxId,  
CORBA::Object\* theObject,  
CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());

**Description** Unregisters a per-object service context handler with Orbix by removing it from the list of Service Context Handlers internally stored.

#### **Parameters**

CtxHandler The handler handling this Service Context ID will be removed.  
theObject The object to unregister the handler for.

**Return Value** 1 (TRUE) if successful; 0 (FALSE) otherwise.

**See Also** ServiceContextHandler class.

### **CORBA::ORB::unregisterPerRequestServiceContext Handler()**

**Synopsis** CORBA::Boolean  
CORBA::ORB::unregisterPerRequestServiceContextHandler(  
CORBA::ULong& CtxId,  
CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());

**Description** Unregisters a per-request service context handler with Orbix by removing it from the list of Service Context Handlers internally stored.

---

**Parameters**

`CtxHandler` The handler handling this Service Context ID will be removed.

**Return Value** 1 (TRUE) if successful; 0 (FALSE) otherwise.

**See Also** `ServiceContextHandler` class.

**CORBA::ORB::useHostNameInIOR()****Synopsis**

```
void CORBA::ORB::useHostNameInIOR(
 CORBA::Boolean val,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description**

When a reference to a remote object enters a client's or server's address space, Orbix constructs a proxy for that object. This proxy (a normal C++ object) is constructed to execute the proxy code corresponding to the actual interface of the true object it represents. Hence if a server object has an operation of the form:

```
// IDL
```

```
// In some interface.
```

```
void op(in Account petal);
```

and if a reference to a (remote) `CurrentAccount` (a derived interface of `Account`) is passed as a parameter to this operation, Orbix tries to set up a proxy for a `CurrentAccount` in the server's address space.

If the server was not linked with the IDL compiler generated proxy code for `CurrentAccount`, Orbix can contact the call's originator to obtain more information and allow the creation of the `CurrentAccount` object.

If this is not acceptable, you should call `useRemoteIsACalls()` on the `CORBA::Orbix` object, passing 0 (FALSE) for the first parameter. The default setting is 1 (TRUE). Setting the value to 0 (FALSE) means that Orbix raises an exception if the server or client is not linked with the actual (dynamic) proxy code.

**See Also**

```
CORBA::ORB::makeIOR()
CORBA::ORB::useTransientPort()
```

### **CORBA::ORB::useRemoteIsACalls()**

**Synopsis**

```
void CORBA::ORB::useRemoteIsACalls(
 CORBA::Boolean useRemoteCall,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description**

Sets the `remoteIsACalls` flag to either disable or enable calls to the operation `CORBA::Object::_is_a()` on a remote object. (This may be necessary to cut down on network traffic.)

**Parameters**

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>useRemoteCall</code> | 1 (TRUE) if the ORB uses calls to the <code>CORBA::Object::_is_a()</code> function on a remote object to determine if a local object is actually a derived instance of the requested type. This is the default value.<br><br>0 (FALSE) if use of the <code>CORBA::Object::_is_a()</code> function on a remote object is explicitly excluded. The ORB raises an exception if it has not been linked with the IDL compiler generated proxy code for all required interfaces. |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Notes**

Orbix specific.

**See Also**

```
CORBA::Object::_is_a()
CORBA::ORB::usingRemoteIsACalls()
CORBA::ORB::resortToStatic()
```

### **CORBA::ORB::useReverseLookup**

**Synopsis**

```
static void CORBA::ORB::useReverseLookup(CORBA::Boolean enable);
```

**Description**

By default an Orbix server will do a reverse lookup of the client machine's host name from its IP address when establishing a new connection. This API can be used to change this behavior. Reverse hostname lookups for clients running on machines not registered with the local network naming service (DNS, NDS, DHCP, WINS or whatever) can take a long time to complete. Thus, this API can be used to speed up the operation of applications in these circumstances. Also, reverse lookups may be unwanted in clustering or multi-homed environments.

**Parameters**

enable            Whether or not reverse lookups are allowed (TRUE by default).

**Notes**            Orbix specific.

**CORBA::ORB::useServiceContexts()****Synopsis**

```
void CORBA::ORB::useServiceContexts(
 CORBA::Boolean useServiceContexts,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description**

Enables or disables Service Context Handlers in Orbix.

**Parameters**

enable            The flag specifying whether they should be enabled (1) or disabled (0).

**See Also**

ServiceContextHandler class.

**CORBA::ORB::useTransientPort()****Synopsis**

```
CORBA::Boolean CORBA::ORB::useTransientPort(
 CORBA::Boolean turnOn);
```

**Description**

By default, an IOR generated by a server will contain the port number for the Orbix daemon. Calling this function with a value of 1 (TRUE) causes any IORs generated by the server to contain the port number for the server itself.

If `useTransientPort()` is set to 1 (TRUE), the function `impl_is_ready()` must be called prior to generating the IOR (`object_to_string`) for the resulting IOR to contain meaningful port number information.

**Parameters**

1 (TRUE)          IOR's generated by the server contain the server's own port number.

0 (FALSE) IOR's generated by the server contain the Orbix daemon's port number. This is the default behavior.

**Return Value** Returns the previous settings.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::useTransientPort()`

### **CORBA::ORB::useTransientPort()**

**Synopsis** `CORBA::Boolean CORBA::ORB::useTransientPort() const;`

**Description** Returns the current setting of the `useTransientPort` flag.

**Return Value**

1 (TRUE) IOR's generated by the server contain the server's own port number instead of the Orbix daemon's port number.

0 (FALSE) IOR's generated by the server contain the Orbix daemon's port number. This is the default behavior.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::useTransientPort(CORBA::Boolean)`

### **CORBA::ORB::usingRemoteIsACalls()**

**Synopsis** `CORBA::Boolean CORBA::ORB::usingRemoteIsACalls(  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())  
const;`

**Description** Returns the current setting of the `remoteIsACalls` flag.

**Return Value**

1 (TRUE) An Orbix process can use `remote_is_a()` calls. This is the default.

---

0 (FALSE)                      Use of calls to the operation  
CORBA::Object::\_is\_a() on a remote  
object is not allowed.

**Notes**                      Orbix specific.

**See Also**                      CORBA::Object::\_is\_a()  
CORBA::ORB::useRemoteIsACalls()  
CORBA::ORB::resortToStatic()

### **CORBA::ORB::usingReverseLookup**

**Synopsis**                      `static CORBA::Boolean CORBA::ORB::usingReverseLookup();`

**Description**                  Returns the current setting of the reverse lookup flag.

**Return Value**                1 (TRUE) when reverse lookups are used, 0 (FALSE) when they are not.

**Notes**                      Orbix specific.

**See Also**                      CORBA::ORB::useReverseLookup

### **CORBA::ORB::usingServiceContexts()**

**Synopsis**                      `CORBA::Boolean CORBA::ORB::usingServiceContexts(  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())  
const;`

**Description**                  Returns whether Service Context Handlers are enabled or not.

**Return Value**

1                      Service Context Handlers are enabled.  
0                      Service Context Handlers are disabled.

**See Also**                      ServiceContextHandler class.





# CORBA::Principal

**Synopsis** Class CORBA::Principal implements the IDL pseudo interface Principal, which represents information about principals (end-users). This information may be used to provide authentication and access control.

**CORBA** // Pseudo IDL  
pseudo interface Principal {};

**Orbix** // C++  
class Principal {  
public:  
 static Principal\_ptr IT\_create(  
 const char\* obj,  
 CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());  
  
 static Principal\_ptr \_duplicate(  
 Principal\_ptr obj,  
 CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());  
  
 static Principal\_ptr \_nil(  
 CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());  
};

**Notes** CORBA compliant.

**See Also** CORBA::BOA::get\_principal()

## CORBA::Principal::Principal()

**Synopsis** CORBA::Principal::Principal();

**Description** Default constructor.

**Notes** Orbix specific.

**See Also** CORBA::Principal::IT\_create()

### **CORBA::Principal::\_duplicate()**

- Synopsis**        `static CORBA::Principal_ptr CORBA::Principal::_duplicate(  
                  CORBA::Principal_ptr obj,  
                  CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description**    Increments the reference count of obj.
- Return Value**   Returns a reference to self.
- Notes**         CORBA compliant.
- See Also**       `CORBA::release()`

### **CORBA::Principal::\_nil()**

- Synopsis**        `static CORBA::Principal_ptr CORBA::Principal::_nil(  
                  CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description**    Returns a nil object reference for a Principal object.
- Notes**         CORBA compliant.
- See Also**       `CORBA::is_nil()`

### **CORBA::Principal::IT\_create()**

- Synopsis**        `static CORBA::Principal_ptr CORBA::Principal::IT_create(  
                  const char* obj,  
                  CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description**    In the absence of a CORBA specified way to create a Principal pseudo object in the current standard C++ mapping, Orbix provides the `IT_create()` function to initialise an object reference for a Principal.
- Use of this function is recommended in preference to C++ operator `new` to ensure memory management consistency.
- Notes**         Orbix specific.

# CORBA::Request

## Synopsis

Class `CORBA::Request` supports the Dynamic Invocation Interface (DII), whereby an application may issue a request for any interface, even if that interface was unknown at the time the application was compiled.

Orbix allows invocations, which are instances of class `CORBA::Request`, to be constructed by specifying at runtime the target object reference, the operation name and the parameters. Such calls are termed “dynamic” because the IDL interfaces used by a program do not have to be “statically” determined at the time the program is designed and implemented.

## CORBA

```
// IDL
pseudo interface Request {
 readonly attribute Object target;
 readonly attribute Identifier operation;
 readonly attribute NVList arguments;
 readonly attribute NamedValue result;
 readonly attribute Environment env;

 attribute Context ctx;

 Status invoke();
 Status send_oneway();
 Status send_deferred();
 Status get_response();
 boolean poll_response();
};
```

## Orbix

```
// C++
class ITDECLSPEC Request : public IT_PseudoIDL {

public:

 ORBStatus invoke();

 ORBStatus invoke(
 Environment& IT_env =
 IT_chooseDefaultEnv());
```

```
ORBStatus send(
 Environment& IT_env =
 IT_chooseDefaultEnv());

ORBStatus get_response(
 Environment& IT_env =
 IT_chooseDefaultEnv());

Object_ptr target(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

const char* operation(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

NVList_ptr arguments(
 Environment& IT_env =
 IT_chooseDefaultEnv());

NamedValue_ptr result(
 Environment& IT_env =
 IT_chooseDefaultEnv());

Environment_ptr env(
 Environment& IT_env =
 IT_chooseDefaultEnv());

void ctx(
 Context_ptr IT_cp,
 Environment& IT_env =
 IT_chooseDefaultEnv());

Context_ptr ctx(
 Environment& IT_env =
 IT_chooseDefaultEnv()) const;

ORBStatus send_oneway(
 Environment& IT_env =
 IT_chooseDefaultEnv());

ORBStatus send_deferred(
 Environment& IT_env =
```

```
 IT_chooseDefaultEnv());

Boolean poll_response(
 Environment& IT_env =
 IT_chooseDefaultEnv());

void set_return_type(
 TypeCode_ptr tc,
 Environment& IT_env =
 IT_chooseDefaultEnv());

void assumeResultOwnership(
 Boolean val);

void assumeArgsOwnership(
 Boolean val);

Request();

Request(
 Object_ptr target,
 const char* operation,
 Boolean isOneWay);

virtual ~Request();

const char* getOperation() const;

void reset(
 Object_ptr obj = 0,
 const char* OperationName = 0,
 Environment& IT_env =
 IT_chooseDefaultEnv());

void reset(
 const char* OperationName);

void setOperation(
 const char* opname);

void setTarget(
 Object_ptr target);
```

```
int descriptor() const;

Request& operator << (const int& i);
Request& operator << (const unsigned int& i);
Request& operator << (const Boolean& o);
Request& operator << (const Short& s);
Request& operator << (const Long& l);
Request& operator << (const UShort& us);
Request& operator << (const ULong& ul);
Request& operator << (const Float& f);
Request& operator << (const Double& d);
Request& operator << (const LongLong& ll);
Request& operator << (const ULongLong& ull);
Request& operator << (const Char& s);
Request& operator << (char*& s);
Request& operator << (const char* s);
Request& operator << (const Context& c);
Request& operator << (const Flags& f);
Request& operator << (Object* const&);
Request& operator << (Any& a);
Request& operator << (TypeCode_ptr& rpTC);
Request& operator << (const IT_FixedBase& f);

Request& insertOctet(const Octet&);
Request& insertOctet(Octet&);

Request& operator >> (Short& s);
Request& operator >> (Long& l);
Request& operator >> (UShort& us);
Request& operator >> (ULong& ul);
Request& operator >> (Float& f);
Request& operator >> (Double& d);
Request& operator >> (LongLong& ll);
Request& operator >> (ULongLong& ull);
Request& operator >> (Char& c);
Request& operator >> (Boolean& o);
Request& operator >> (char*& s);
Request& operator >> (Object_ptr&);
Request& operator >> (Any&);
Request& operator >> (TypeCode_ptr& rpTC);
Request& operator >> (IT_FixedBase& f);

Request& extractOctet(Octet&);
```

```
void encodeStringOp(const char* s, ULong bnd = 0);
void decodeStringOp(char*& s);
void decodeBndStrOp(char*& s, Long bnd);
void decodeInOutStrOp(char*& s, Long bnd);
void encode(Principal_ptr);
void decode(Principal_ptr&);
void decodeInOut(Principal_ptr&);
void encode(TypeCode_ptr);
void decode(TypeCode_ptr&);
void decodeInOut(TypeCode_ptr&);
void encode(NamedValue_ptr);
void decode(NamedValue_ptr&);
void decodeInOut(NamedValue_ptr&);
void encodeCharArray(char*& s, ULong len);
void decodeCharArray(char*& s, ULong& len);
void encodeUCharArray(unsigned char*& s, ULong len);
void decodeUCharArray(unsigned char*& s, ULong& len);
void encodeShortArray(Short*& s, ULong len);
void decodeShortArray(Short*& s, ULong& len);
void encodeUShortArray(UShort*& s, ULong len);
void decodeUShortArray(UShort*& s, ULong& len);
void encodeLongArray(Long*& s, ULong len);
void decodeLongArray(Long*& s, ULong& len);
void encodeULongArray(ULong*& s, ULong len);
void decodeULongArray(ULong*& s, ULong& len);
void encodeLongLongArray(LongLong*& s, ULong len);
void decodeLongLongArray(LongLong*& s, ULong& len);
void encodeULongLongArray(ULongLong*& s, ULong len);
void decodeULongLongArray(ULongLong*& s, ULong& len);
void encodeFloatArray(Float*& s, ULong len);
void decodeFloatArray(Float*& s, ULong& len);
void encodeDoubleArray(Double*& s, ULong len);
void decodeDoubleArray(Double*& s, ULong& len);
void encodeOctetArray(Octet*& s, ULong len);
void decodeOctetArray(Octet*& s, ULong& len);
void encodeBooleanArray(Boolean*& s, ULong len);
void decodeBooleanArray(Boolean*& s, ULong& len);
void encodeFixedArray(IT_FixedBase*& f, ULong len);
void decodeFixedArray(IT_FixedBase*& f, ULong& len);

static Request_ptr IT_create(
 Environment& IT_env =
```

```
 IT_chooseDefaultEnv());

static Request_ptr IT_create(
 Object_ptr target,
 const char* OperationName = 0,
 Environment& IT_env =
 IT_chooseDefaultEnv(),
 Boolean x = 0,
 Boolean y = 0);

static Request_ptr _duplicate(
 Request_ptr obj,
 Environment& IT_env =
 IT_chooseDefaultEnv());

static Request_ptr _nil(
 Environment& IT_env =
 IT_chooseDefaultEnv());

};
```

**Notes** CORBA compliant.

### **CORBA::Request::Request()**

**Synopsis** CORBA::Request::Request();

**Description** Default constructor. The target object and the operation name for the request should then be specified.

**Notes** Orbix specific. Refer to CORBA::Object::\_create\_request() and CORBA::Object::\_request() for CORBA compliant ways of constructing a Request.

**See Also** CORBA::Object::\_create\_request()  
CORBA::Object::\_request()  
CORBA::Request::IT\_create()  
Other Request constructor.



**CORBA::Request::Request()**

- Synopsis** `CORBA::Request::Request(CORBA::Object_ptr target,  
const Identifier OperationName = 0,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Constructs a Request. A request is built by specifying its target object's reference.
- Parameters**
- |                            |                                                                                                                        |
|----------------------------|------------------------------------------------------------------------------------------------------------------------|
| <code>target</code>        | The object that is the target of the request.                                                                          |
| <code>OperationName</code> | The operation name for the request. If not set here, it may be set using <code>CORBA::Request::setOperation()</code> . |
- Notes** Orbix specific. Refer to `CORBA::Object::_create_request()` and `CORBA::Object::_request()` for CORBA compliant ways of constructing a Request.
- See Also** `CORBA::Object::_create_request()`  
`CORBA::Object::_request()`  
`CORBA::Request::setOperation()`  
`CORBA::Request::IT_create()`  
Other Request constructor.

**CORBA::Request::~~Request()**

- Synopsis** `virtual CORBA::Request::~~Request();`
- Description** Destructor.
- Notes** Orbix specific.

**CORBA::Request::addToContextList()**

- Synopsis** `CORBA::Boolean CORBA::Request::addToContextList(  
CORBA::ServiceContext* context);`
- Description** Adds the Service Context that is passed in to the Service Context List stored in the request.

### Parameters

context                    The Service Context to be added.

**Return Value**    Always TRUE.

**See Also**        ServiceContextHandler class.

### **CORBA::Request::getContextList()**

**Synopsis**        CORBA::ServiceContextList\* CORBA::Request::getContextList(  
                  CORBA::Environment& IT\_env = CORBA::IT\_chooseDefaultEnv());

**Description**    Obtains the Service Context List from the request.

**Return Value**    The list of service contexts as held by the request.

**See Also**        ServiceContextHandler class.

### **CORBA::Request::operator>>()**

**Synopsis**        Request& operator>>(Short&);  
Request& operator>>(Long&);  
Request& operator>>(LongLong&);  
Request& operator>>(UShort&);  
Request& operator>>(ULong&);  
Request& operator>>(ULongLong&);  
Request& operator>>(Float&);  
Request& operator>>(Double&);  
Request& operator>>(Char&);  
Request& operator>>(Boolean&);  
Request& operator>>(char\*&);  
Request& operator>>(Object\_ptr&);  
Request& operator>>(Any&);  
Request& operator>>(IT\_FixedBase&);

**Description**    Once an invocation has been made, you can examine the operation's return value using the extraction operator, `operator>>()`. If there are any `out` and `inout` parameters, these parameters are modified by the call, and no special action is required to access their values.

To extract a CORBA::Octet from a Request, the function CORBA::Request::extractOctet() may be used. To extract a user-defined type, Refer to CORBA::extract(). To extract an array, Refer to CORBA::Request::decodeArray().

**Return Value** Returns a reference to self.

**Notes** Orbix specific. The CORBA compliant function is CORBA::Request::result().

**See Also** CORBA::Request::result()  
CORBA::Request::extractOctet()  
CORBA::Request::decodeArray()  
CORBA::extract()

## CORBA::Request::operator<<()

### Synopsis

```
CORBA::Request& operator<<(const CORBA::Boolean&);
CORBA::Request& operator<<(const CORBA::Short&);
CORBA::Request& operator<<(const CORBA::Long&);
CORBA::Request& operator<<(const CORBA::LongLong&);
CORBA::Request& operator<<(const CORBA::UShort&);
CORBA::Request& operator<<(const CORBA::ULong&);
CORBA::Request& operator<<(const CORBA::ULongLong&);
CORBA::Request& operator<<(const CORBA::Float&);
CORBA::Request& operator<<(const CORBA::Double&);
CORBA::Request& operator<<(const CORBA::Char&);
CORBA::Request& operator<<(const char*&);
CORBA::Request& operator<<(const CORBA::Context&);
CORBA::Request& operator<<(const CORBA::Flags&);
CORBA::Request& operator<<(CORBA::Object_ptr const&);
CORBA::Request& operator<<(CORBA::Any&);
CORBA::Request& operator<<(CORBA::IT_FixedBase&);
```

### Description

The insertion operator, operator<<(), may be used to insert the parameters into a Request. The parameters must be inserted in the correct order and each parameter must be passed with its correct mode (otherwise the dynamic type checking will fail). The default mode is inMode. The manipulators inMode, outMode, and inoutMode affect *all subsequent* uses of operator<<() on a given Request until the next mode change.

Input (in) parameters are not copied into the request argument list; thus, if the values of the variables are changed between being inserted and the invocation being made, the new values are transmitted (this is done to adhere to the CORBA specification). In other words, `operator<<()` uses “call by reference” semantics, and you must be careful to ensure that the parameters remain in existence and have the desired values when the invocation of the `Request` is actually made. An example of an error is to insert a local variable within a function and to return from the function before the `Request` invocation is made.

An example of the use of `operator<<()` is:

```
// IDL
long Foo(in long l, inout float f, out char c);
```

Parameters can be inserted as follows:

```
// C++
CORBA::Long l = 4L;
CORBA::Float fl = 8.9;
char ch;
// r is an object of type CORBA::Request.
r << l
 << CORBA::inoutMode << fl
 << CORBA::outMode << ch;
```

The parameters to a request are dynamically type checked by Orbix on the server's node, when the request arrives at the remote object.

Parameters inserted using `operator<<()` are, by default nameless. The name of a parameter can be given explicitly using `CORBA::arg()`:

```
// C++
// Insert parameter "height"
r << CORBA::arg("height") << 65;
```

The naming of parameters does not remove the requirement that parameters must be inserted in the proper order. However, if the same parameter name is used again, its previous value is replaced with the new value.

Note that `arg()` affects only the *next* use of `operator<<()`.

To insert a `CORBA::Octet` into a `Request`, the function `Request::insertOctet()` must be used. To insert a user-defined type, see `CORBA::insert()`. To insert an array see `CORBA::Request::encodeArray()`.

**Return Value** Returns a reference to self.

**Notes** Orbix specific. Refer to `CORBA::Request::arguments()` for CORBA compliant ways of constructing a `Request`.

**See Also** `CORBA::Request::insertOctet()`  
`CORBA::Request::encodeArray()`  
`CORBA::insert()`  
`CORBA::arg()`

### **CORBA::Request::\_duplicate()**

**Synopsis**

```
static CORBA::Request_ptr CORBA::Request::_duplicate(
 CORBA::Request_ptr obj,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Increments the reference count of `obj`.

**Return Value** Returns a reference to self.

**Notes** Orbix specific.

**See Also** `CORBA::release()`

### **CORBA::Request::\_nil()**

**Synopsis**

```
static CORBA::Request_ptr CORBA::Request::_nil(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Returns a nil object reference for a `Request`.

**Notes** Orbix specific.

**See Also** `CORBA::is_nil()`

### **CORBA::Request::arguments()**

**Synopsis**

```
CORBA::NVList_ptr CORBA::Request::arguments(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description** Returns the arguments to the `Request`'s operation in an `NVList`.

**Notes** CORBA compliant.

**See Also** `CORBA::NVList`

### **CORBA::Request::assumeArgsOwnership()**

|                    |                                                                                                                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <code>void CORBA::Request::assumeArgsOwnership(CORBA::Boolean val);</code>                                                                                                                                                                                                                                  |
| <b>Description</b> | Specifies whether a <code>CORBA::Request</code> object should assume responsibility for the memory associated with the <code>arg_list</code> parameter passed to <code>CORBA::Object::_create_request()</code> . By default, a <code>CORBA::Request</code> object does not assume ownership of this memory. |
| <b>Parameters</b>  | <p><code>val</code> A non-zero (<code>TRUE</code>) value indicates that the <code>CORBA::Request</code> object should assume ownership of the <code>arg_list</code> parameter. A zero (<code>FALSE</code>) value indicates that you should manage the memory for this parameter.</p>                        |
| <b>Notes</b>       | CORBA compliant.                                                                                                                                                                                                                                                                                            |
| <b>See Also</b>    | <code>CORBA::Object::_create_request()</code>                                                                                                                                                                                                                                                               |

### **CORBA::Request::assumeResultOwnership()**

|                    |                                                                                                                                                                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <code>void CORBA::Request::assumeResultOwnership(CORBA::Boolean val);</code>                                                                                                                                                                                                                              |
| <b>Description</b> | Specifies whether a <code>CORBA::Request</code> object should assume responsibility for the memory associated with the <code>result</code> parameter passed to <code>CORBA::Object::_create_request()</code> . By default, a <code>CORBA::Request</code> object does not assume ownership of this memory. |
| <b>Parameters</b>  | <p><code>val</code> A non-zero (<code>TRUE</code>) value indicates that the <code>CORBA::Request</code> object should assume ownership of the <code>result</code> parameter. A zero (<code>FALSE</code>) value indicates that the programmer should manage the memory for this parameter.</p>             |
| <b>Notes</b>       | CORBA compliant.                                                                                                                                                                                                                                                                                          |
| <b>See Also</b>    | <code>CORBA::Object::_create_request()</code>                                                                                                                                                                                                                                                             |

**CORBA::Request::ctx()**

- Synopsis** `CORBA::Context_ptr CORBA::Request::ctx(  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())  
const;`
- Description** Gets the Context associated with a request.
- Notes** CORBA compliant.
- See Also** `CORBA::Request::ctx(CORBA::Context_ptr c)`  
`CORBA::Context`

**CORBA::Request::ctx()**

- Synopsis** `void CORBA::Request::ctx(CORBA::Context_ptr c,  
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Inserts a Context into a request.
- Notes** CORBA compliant.
- See Also** `CORBA::Request::ctx(CORBA::Context_ptr c);`  
`CORBA::Context`

**CORBA::Request::decodeArray()**

- Synopsis** `void decodeCharArray(char*&, CORBA::ULong& len);  
void decodeUCharArray(unsigned char*&, CORBA::ULong& len);  
void decodeShortArray(CORBA::Short*&, CORBA::ULong& len);  
void decodeUShortArray(CORBA::UShort*&, CORBA::ULong& len);  
void decodeLongArray(CORBA::Long*&, CORBA::ULong& len);  
void decodeULongArray(CORBA::ULong*&, CORBA::ULong& len);  
void decodeLongLongArray(CORBA::LongLong*&, ULong& len);  
void decodeULongLongArray(CORBA::ULongLong*&, CORBA::ULong& len);  
void decodeFloatArray(CORBA::Float*&, CORBA::ULong& len);  
void decodeDoubleArray(CORBA::Double*&, CORBA::ULong& len);  
void decodeOctetArray(CORBA::Octet*&, CORBA::ULong& len);  
void decodeBooleanArray(CORBA::Boolean*&, CORBA::ULong& len);  
void decodeFixedArray(CORBA::IT_FixedBase*&, CORBA::ULong& len);`

**Description** Once an invocation has been made, you can examine the operation's return value. These functions allow a return value which is an array of basic types to be extracted from a `Request`. If there are any `out` and `inout` parameters, these parameters are modified by the call, and no special action is required to access their values.

**Parameters**

`len` Contains the length of the array (after the call).

**Notes** Orbix specific. The CORBA compliant function is `CORBA::Request::result()`.

**See Also**

`CORBA::Request::result()`  
`CORBA::Request::encodeArray()`  
`CORBA::Request::operator>>()`  
`CORBA::Request::extractOctet()`  
`CORBA::extract()`

### **CORBA::Request::decodeStringOp()**

**Synopsis**

```
void CORBA::Request::decodeStringOp(char*& s);
```

**Description**

This functions allows an unbounded string to be retrieved from a `Request`. Parameters must be retrieved in the correct order.

**Parameters**

`s` The location of the retrieved, decoded null terminated string.

**Notes**

Orbix specific.

**See Also**

`CORBA::Request::encodeStringOp()`  
`CORBA::Request::decodeBndStrOp()`  
`CORBA::Request::decodeInOutStrOp()`



**CORBA::Request::decodeBndStrOp()****Synopsis**

```
void CORBA::Request::decodeBndStrOp(
 char*& s, CORBA::Long bnd);
```

**Description**

This functions allows an bounded string to be retrieved from a Request. Parameters must be retrieved in the correct order.

**Parameters**

- s The location of the retrieved, decoded, null terminated string.
- bnd The length of the retrieved, decoded string not including the terminating null character.

**Notes**

Orbix specific.

**See Also**

```
CORBA::Request::encodeStringOp()
CORBA::Request::decodeStringOp()
CORBA::Request::decodeInOutStrOp()
```

**CORBA::Request::decodeInOutStrOp()****Synopsis**

```
void CORBA::Request::decodeInOutStrOp(
 char*& s, CORBA::Long bnd);
```

**Description**

This functions allows a string which is being used as an in out parameter to an operation to be retrieved from a Request. Parameters must be retrieved in the correct order.

**Parameters**

- op The location of the retrieved, decoded, null terminated string. bnd The length of the retrieved, decoded string not including the terminating null character.
- bnd The length of the retrieved, decoded string not including the terminating null character.

**Notes**

Orbix specific.

**See Also**

```
CORBA::Request::encodeStringOp()
CORBA::Request::decodeStringOp()
CORBA::Request::decodeBndStrOp()
```

### **CORBA::Request::descriptor()**

**Synopsis**      `int CORBA::Request::descriptor(void) const;`

**Description**      Returns the file descriptor associated with a request.

**Notes**      Orbix specific.

### **CORBA::Request::encodeArray()**

**Synopsis**

```
void encodeCharArray(char*&, CORBA::ULong len);
void encodeUCharArray(unsigned char*&, CORBA::ULong len);
void encodeShortArray(CORBA::Short*&, CORBA::ULong len);
void encodeUShortArray(CORBA::UShort*&, CORBA::ULong len);
void encodeLongArray(CORBA::Long*&, CORBA::ULong len);
void encodeULongArray(CORBA::ULong*&, CORBA::ULong len);
void encodeLongLongArray(CORBA::LongLong*&, CORBA::ULong& len);
void encodeULongLongArray(CORBA::ULongLong*&, CORBA::ULong& len);
void encodeFloatArray(CORBA::Float*&, CORBA::ULong len);
void encodeDoubleArray(CORBA::Double*&, CORBA::ULong len);
void encodeOctetArray(CORBA::Octet*&, CORBA::ULong len);
void encodeBooleanArray(CORBA::Boolean*&, CORBA::ULong len);
void encodeFixedArray(CORBA::IT_FixedBase*&, CORBA::ULong& len);
```

**Description**      These functions allow an array of basic types to be inserted into a `Request` as a parameter to an operation. Parameters must be inserted in the correct order.

#### **Parameters**

`len`    The length of the array.

**Notes**      Orbix specific.

**See Also**

```
CORBA::Request::decodeArray()
CORBA::Request::operator<<()
CORBA::Request::insertOctet()
CORBA::insert()
```

## CORBA::Request::encodeStringOp()

**Synopsis**

```
void CORBA::Request::encodeStringOp(
 const char* op, CORBA::ULong bnd = 0);
```

**Description**

This functions allows a string to be inserted into a Request as a parameter to an operation. Parameters must be inserted in the correct order.

**Paramaters**

`op` The string to be encoded.

`bnd` The bound which must be either equal to, or greater than, the length of the string to be encoded. A value of 0 bytes indicates an unbounded string.

**Notes**

Orbix specific.

**See Also**

```
CORBA::Request::decodeStringOp()
CORBA::Request::decodeBndStrOp()
CORBA::Request::decodeInOutStrOp()
```

## CORBA::Request::env()

**Synopsis**

```
CORBA::Environment_ptr CORBA::Request::env(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

**Description**

Returns the `Environment` associated with the request from which exceptions raised in DII calls can be accessed.

**Notes**

CORBA compliant.

**See Also**

```
CORBA::Environment()
```

## CORBA::Request::extractOctet()

**Synopsis**

```
CORBA::Request& CORBA::Request::extractOctet(CORBA::Octet&);
```

**Description**

Once an invocation has been made, you can examine the operation's return value. If there are any `out` and `inout` parameters, these parameters are modified by the call, and no special action is required to access their values. This function extracts a return value of type `octet` from a request.

The function `extractOctet()` is provided, rather than `operator<<()` because `octet` has the same C++ type as for IDL `boolean`.

**Notes** Orbix specific. The CORBA compliant function is `CORBA::Request::result()`.

**See Also** `CORBA::Request::insertOctet()`  
`CORBA::Request::operator>>()`  
`CORBA::Request::decodeArray()`  
`CORBA::Request::result()`  
`CORBA::extract()`

### **CORBA::Request::get\_response()**

**Synopsis** `CORBA::Status CORBA::Request::get_response();`

**Description** Determines whether a request has completed successfully. It returns only when the request (invoked using `send_deferred()`) has completed. If return value indicates success, the `out` and `inout` parameters and return values defined in the `Request` are valid.

**Return Value** Returns 1 (TRUE) if the `Request` completed successfully; returns 0 (FALSE) otherwise.

**Notes** CORBA compliant.

**See Also** `CORBA::Request::result()`  
`CORBA::Request::send_deferred()`

### **CORBA::Request::insertOctet()**

**Synopsis** `CORBA::Request& CORBA::Request::insertOctet(  
const CORBA::Octet& o);`

**Description** Inserts a parameter of type `octet` into a request. The parameters must be inserted in the correct order.

The function `insertOctet()` is provided, rather than `operator<<()` because `octet` has the same C++ type as for IDL `boolean`.

**Notes** Orbix specific.

**See Also** `CORBA::Request::operator<<()`  
`CORBA::Request::encodeArray()`

```
CORBA::insert()
CORBA::Request::extractOctet()
```

## CORBA::Request::invoke()

- Synopsis** `CORBA::Status CORBA::Request::invoke();`
- Description** Instructs Orbix to make a request. The parameters to the request must already be set up. The caller is blocked until the request has been processed by the target object or an exception occurs.
- To make a non-blocking request, see `CORBA::Request::send_deferred()` and `CORBA::Request::send_oneway()`.
- Return Value** Returns 1 (TRUE) if successful, 0 (FALSE) otherwise.
- Notes** CORBA compliant.
- See Also** `CORBA::Request::send_oneway()`  
`CORBA::Request::send_deferred()`  
`CORBA::Request::result()`

## CORBA::Request::IT\_create()

- Synopsis**
- ```
static CORBA::Request_ptr CORBA::Request::IT_create(  
    CORBA::Object_ptr target,  
    const Identifier OperationName = 0,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());  
static CORBA::Request_ptr CORBA::Request::IT_create(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```
- Description** For consistency with other pseudo object types for which there is no CORBA specified way in the current C++ mapping to obtain an object reference, Orbix provides the `IT_create()` function for class `Request`. To ensure memory management consistency, you should not use the C++ `new` operator to create an `Request`.
- Refer to the corresponding constructor for details of the parameters to `IT_create()`.

Notes Orbix specific. Refer to `CORBA::Object::_create_request()` and `CORBA::Object::_request()` for CORBA compliant ways of creating a `Request`.

See Also `CORBA::Object::_create_request()`
`CORBA::Object::_request()`
`Request` constructors.

CORBA::Request::operation()

Synopsis

```
const char* CORBA::Request::operation(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())  
const;
```

Description Gets the `Request`'s operation name.

Notes CORBA compliant.

See Also `CORBA::Request::setOperation()`

CORBA::Request::poll_response()

Synopsis

```
CORBA::Boolean CORBA::REQUEST::poll_response(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description A caller who makes an operation request using `send_deferred()` may call `poll_response()` to determine whether the operation has completed. The function returns immediately. If the operation has completed, the result is available in the `Request`.

Return Value Returns 1 (`TRUE`) if the operation has completed successfully indicating that the return value and `out` and `inout` parameters in the `Request` are valid; returns 0 (`FALSE`) otherwise.

Notes CORBA compliant.

See Also `CORBA::Request::send_oneway()`
`CORBA::Request::send_deferred()`
`CORBA::Request::get_response()`
`CORBA::ORB::poll_next_response()`

CORBA::Request::reset()**Synopsis**

```
void CORBA::Request::reset(const char* operationName);  
void reset(CORBA::Object_ptr obj = 0,  
           const char* OperationName = 0);
```

Description

Allows a Request object to be reused. You can individually reset the target object and the operation name or both.

Parameters

obj The target object. (If set to 0, the target object is not reset.)
operationName The operation name.

Notes

Orbix specific.

CORBA::Request::result()**Synopsis**

```
CORBA::NamedValue_ptr CORBA::Request::result(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description

Returns the result of the operation request in a NamedValue.

Notes

CORBA compliant.

See Also

```
operator>>()  
CORBA::extract()  
CORBA::extractOctet()  
CORBA::decodeArray()
```

CORBA::Request::send_deferred()**Synopsis**

```
CORBA::Status CORBA::Request::send_deferred(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description

Instructs Orbix to make the request. The arguments to the request must already be set up. The caller is not blocked, and thus may continue in parallel with the processing of the call by the target object.

You can use the function `CORBA::poll_response()` to determine whether the operation completed.

You can use the function `CORBA::get_response()` to determine the outcome of the request. Memory leakage results if this function is not called for a request issued with `send_deferred()`.

To make a blocking request, see `CORBA::Request::invoke()`.

Return Value Returns 1 (TRUE) if successful, 0 (FALSE) otherwise.

Notes CORBA compliant.

See Also `CORBA::Request::send_oneway()`
`CORBA::ORB::send_multiple_requests_deferred()`
`CORBA::Request::invoke()`
`CORBA::Request::poll_response()`
`CORBA::Request::get_response()`

CORBA::Request::send_oneway()

Synopsis

```
CORBA::Status CORBA::Request::send_oneway(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description Instructs Orbix to make the oneway request. The arguments to the request must already be set up. The caller is not blocked, and thus may continue in parallel with the processing of the call by the target object.

This function may be used even if the operation has not been defined to be `oneway` in its IDL definition. The caller should not expect any `in` or `inout` parameters to be updated.

To make a blocking request, see `CORBA::Request::invoke()`.

Return Value Returns 1 (TRUE) if successful, 0 (FALSE) otherwise.

Notes CORBA compliant.

See Also `CORBA::Request::send_deferred`
`CORBA::ORB::send_multiple_requests_oneway()`
`CORBA::Request::invoke()`
`CORBA::Request::poll_response()`
`CORBA::Request::get_response()`

CORBA::Request::setOperation()

Synopsis `void CORBA::Request::setOperation(const char* opname);`

Description Sets the operation name for the request.

Notes Orbix specific.

See Also `CORBA::Request::operation()`
 `CORBA::Request::_create_request()`
 `CORBA::Request::_request()`

CORBA::Request::set_return_type()

- Synopsis** `void CORBA::Request::set_return_type(CORBA::TypeCode_ptr tc,
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** When using the DII with the CORBA Internet Inter-ORB Protocol (IIOP), you must set the return type of a request before invoking the request. This function allows you to specify the `TypeCode` associated with a request when setting up a `CORBA::Request` object.
- Parameters**
- `tc` The `TypeCode` for the return type of the operation associated with the `CORBA::Request` object.
- Notes** CORBA compliant.

CORBA::Request::setTarget()

- Synopsis** `void CORBA::Request::setTarget(CORBA::Object_ptr target);`
- Description** Sets the `Request`'s target object. The reference count of `target` is not incremented.
- Notes** Orbix specific.
- See Also** `CORBA::Request::target()`

CORBA::Request::target()

- Synopsis** `Object_ptr target(
 CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
 const;`
- Description** Gets the `Request`'s target object.
- Notes** CORBA compliant.
- See Also** `CORBA::Request::setTarget()`

CORBA::ServerRequest

Synopsis Class `ServerRequest` describes a Dynamic Skeleton Interface (DSI) operation request. It is analogous to the `Request` class used in the Dynamic Invocation Interface (DII).

An instance of `ServerRequest` is created by Orbix when it receives an incoming request that is to be handled by the DSI—that is, an instance of `CORBA::DynamicImplementation` has been registered to handle the target interface.

An instance of `ServerRequest` is a pseudo-object so an instance of a `ServerRequest` cannot be transmitted in an IDL operation.

You should not define derived classes of `ServerRequest`.

CORBA

```
// IDL
pseudo interface ServerRequest {
    Identifier op_name();
    Context ctx();
    attribute any result;1
    void params(inout NVList parms);

    // The following are Orbix specific:
    readonly attribute Object target;
    readonly attribute Identifier operation;
    // Synonym for op_name().
    attribute NVList arguments;
    // Closely related to params()
    attribute any exception;
    attribute Environment env;
};
```

1. The standard specifies this to be an operation; that is, the value can only be read.

Orbix

```
// C++
class ServerRequest {
public:
    virtual const char* op_name(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const = 0;

    virtual Context_ptr ctx(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const = 0;

    virtual void params(NVList_ptr,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) =
0;

    virtual Any* result(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) =
0;

    virtual OperationDef_ptr op_def(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) =
0;

    virtual void result(CORBA::Any*,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) =
0;

    virtual void exception(CORBA::Any*,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) =
0;

    virtual Object_ptr target(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const = 0;

    virtual const char* operation(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const = 0;

    virtual NVList_ptr arguments(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) =
0;
```

```
virtual void arguments (NVList_ptr,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) =  
0;  
  
virtual Environment_ptr env(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) =  
0;  
virtual void env(Environment_ptr,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) =  
0;  
  
protected:  
    ServerRequest();  
    virtual ~ServerRequest();  
    ServerRequest* operator&();  
    const ServerRequest* operator&() const;  
};
```

Notes CORBA compliant.

See Also CORBA::DynamicImplementation

CORBA::ServerRequest::ServerRequest()

Synopsis CORBA::ServerRequest::ServerRequest();

Description Default constructor. The constructor is protected because instances of ServerRequest are intended to be created and destroyed by Orbix.

Notes CORBA compliant.

CORBA::ServerRequest::~~ServerRequest()

Synopsis virtual CORBA::ServerRequest::~~ServerRequest();

Description Destructor. The destructor is protected because instances of ServerRequest are intended to be created and destroyed by Orbix.

Notes CORBA compliant.

CORBA::ServerRequest::arguments()

- Synopsis** `CORBA::ServerRequest::arguments(CORBA::NVList_ptr nvl,
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) = 0;`
- Description** Allows (a redefinition of) `CORBA::DynamicImplementation::invoke()` to specify the values of incoming arguments and to return out and inout arguments. It must be called *exactly* once in each execution of the `invoke()` function.
- Notes** Orbix specific. Added for symmetry with `CORBA::Request`. See `CORBA::ServerRequest::params()` for CORBA compliant version of this function.
- See Also** `CORBA::ServerRequest::params()`
`CORBA::DynamicImplementation::invoke()`

CORBA::ServerRequest::ctx()

- Synopsis** `CORBA::Context_ptr CORBA::ServerRequest::ctx(
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const = 0;`
- Description** Gets the `Context` associated with the call. It can be called at most once. If it is called, it must be called before `CORBA::ServerRequest::params()` or `CORBA::ServerRequest::arguments()`.
- Notes** Orbix specific. Added for symmetry with `CORBA::Request`.
- See Also** `CORBA::Context`

CORBA::ServerRequest::exception()

- Synopsis** `virtual void CORBA::ServerRequest::exception(CORBA::Any*,
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) = 0;`
- Description** Allows the Dynamic Implementation Routine (DIR), that is, a redefinition of `CORBA::DynamicImplementation::invoke()` to return an exception to the caller. In C++, the exception is given as a pointer to a `CORBA::Any`, which holds the exception to be returned to the caller.
- Notes** CORBA compliant.

See Also CORBA::Environment()
CORBA::DynamicImplementation::invoke()

CORBA::ServerRequest::env()

Synopsis CORBA::Environment_ptr CORBA::ServerRequest::env(
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

Description Returns the Environment associated with the call.

Notes Orbix specific. Added for symmetry with CORBA::Request.

See Also CORBA::Environment()

CORBA::ServerRequest::env()

Synopsis virtual void CORBA::ServerRequest::env(CORBA::Environment_ptr,
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) = 0;

Description Sets the Environment associated with the ServerRequest.

Notes Orbix specific.

See Also CORBA::Environment()

CORBA::ServerRequest::op_def()

Synopsis CORBA::OperationDef_ptr CORBA::ServerRequest::op_def(
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const = 0;

Description Returns the Interface Repository object describing the operation being invoked.

Notes CORBA compliant. Use of this function requires the code to be linked with the Interface Repository library.

See Also CORBA::ServerRequest::operation()

CORBA::ServerRequest::op_name()

Synopsis

```
const char* CORBA::ServerRequest::op_name(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())  
const = 0;
```

Description

Gets the name of the operation being invoked.

It must be called at least once in each execution of the Dynamic Implementation Routine (DIR), that is, in each redefinition of `CORBA::DynamicImplementation::invoke()`.

Notes

CORBA compliant.

See Also

`CORBA::ServerRequest::operation()`
`CORBA::DynamicImplementation::invoke()`

CORBA::ServerRequest::operation()

Synopsis

```
const char* CORBA::ServerRequest::operation(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())  
const = 0;
```

Description

Gets the name of the operation being invoked.

It must be called at least once in each execution of the Dynamic Implementation Routine (DIR), that is, in each redefinition of `CORBA::DynamicImplementation::invoke()`.

Notes

Orbix specific. Added for symmetry with `CORBA::Request`. Refer to `CORBA::ServerRequest::op_name()` for CORBA compliant version of this function.

See Also

`CORBA::ServerRequest::op_name()`
`CORBA::DynamicImplementation::invoke()`

CORBA::ServerRequest::params()

- Synopsis** `virtual void CORBA::ServerRequest::params(CORBA::NVList_ptr nvl,
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const = 0;`
- Description** Allows `CORBA::DynamicImplementation::invoke()` to specify the values of incoming arguments and to return out and inout arguments.
- It must be called *exactly* once in each execution of the Dynamic Implementation Routine (DIR), that is, in each redefinition of `CORBA::DynamicImplementation::invoke()` function.
- Parameters**
- `nvl` The argument list.
- Notes** CORBA compliant.
- See Also** `CORBA::ServerRequest::arguments()`

CORBA::ServerRequest::result()

- Synopsis** `CORBA::Any* CORBA::ServerRequest::result(
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()) = 0;`
- Description** Allows the `CORBA::DynamicImplementation::invoke()` operation to return the result of an operation request in a `CORBA::Any`.
- Must be called once for operations with non-void return types and not at all for operations with void return types. If it is called, `CORBA::ServerRequest::exception()` cannot be used.
- Notes** Orbix specific.
- See Also** `CORBA::ServerRequest::exception()`

CORBA::ServerRequest::target()

Synopsis

```
CORBA::Object_ptr CORBA::ServerRequest::target(  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv()  
    const = 0;
```

Description

Returns an object reference to the target object. The target object does not really exist as a normal CORBA object so this is an object (of a derived type of `CORBA::Object`) that is created by Orbix temporarily for the duration of the call.

Notes

Orbix specific.

CORBA::ServiceContextHandler

Synopsis

Class ServiceContextHandler is an abstract class that describes the interface to service context handlers. To implement a Service Context Handler you define a derived class of the ServiceContextHandler class and redefine some or all of the handler functions as described in the Orbix C++ Programmer's Guide.

Orbix

```
// C++
class ServiceContextHandler {
public:
    ServiceContextHandler(CORBA::ULong ContextId,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
    virtual ~ServiceContextHandler();
    virtual CORBA::Boolean incomingRequest(
        CORBA::Request& req,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
    virtual CORBA::Boolean outboundRequest(
        CORBA::Request& req,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
    virtual CORBA::Boolean incomingReply(
        CORBA::Request& req,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
    virtual CORBA::Boolean outboundReply(
        CORBA::Request& req,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
    virtual CORBA::ULong context_id();
};
```

CORBA::ServiceContextHandler::ServiceContextHandler()

Synopsis

```
CORBA::ServiceContextHandler::ServiceContextHandler(
    CORBA::ULong ContextId,
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description

Constructor. Should be called by subclasses to initialize the Context ID.

Parameters

ContextId The ID of the service contexts handled by this handler.

CORBA::ServiceContextHandler::~~ServiceContextHandler()

Synopsis

```
virtual CORBA::ServiceContextHandler::~ServiceContextHandler();
```

Description

Destructor. Derived classes may need to redefine the destructor.

CORBA::ServiceContextHandler::incomingRequest()

Synopsis

```
virtual CORBA::Boolean  
CORBA::ServiceContextHandler::incomingRequest(  
    CORBA::Request& req,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description

Defines the service context handler to be called when a request arrives in the server address space.

If not redefined in a derived class, the following implementation is inherited:

```
// C++  
{ return 1; } // no error
```

Parameters

req The current request that may be modified to carry a service request, or inspected to look for existing service contexts.

Return Value

- 1 (TRUE) Signals success of the handler to Orbix.
- 0 (FALSE) Signals an error in the handler to Orbix. Orbix will display a diagnostic message in this case. The message will not be displayed if diagnostics are switched off.

CORBA::ServiceContextHandler::outboundRequest()

Synopsis

```
virtual CORBA::Boolean  
CORBA::ServiceContextHandler::outboundRequest(  
    CORBA::Request& req,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description

Defines the service context handler to be called when a request leaves the client's address space.

If not redefined in a derived class, the following implementation is inherited:

```
// C++  
{ return 1; } // no error
```

Parameters

`req` The current request that may be modified to carry a service request, or inspected to look for existing service contexts.

Return Value

- 1 (TRUE) Signals success of the handler to Orbix.
- 0 (FALSE) Signals an error in the handler to Orbix. Orbix will display a diagnostic message in this case. The message will not be displayed if diagnostics are switched off.

CORBA::ServiceContextHandler::incomingReply()

Synopsis

```
virtual CORBA::Boolean  
CORBA::ServiceContextHandler::incomingReply(  
    CORBA::Request& req,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
```

Description

Defines the service context handler to be called when a reply enters the client's address space.

If not redefined in a derived class, the following implementation is inherited:

```
// C++  
{ return 1; } // no error
```

Parameters

`req` The current request that may be modified to carry a service request, or inspected to look for existing service contexts.

Return Value

- 1 (TRUE) Signals success of the handler to Orbix.
- 0 (FALSE) Signals an error in the handler to Orbix. Orbix will display a diagnostic message in this case. The message will not be displayed if diagnostics are switched off.

CORBA::ServiceContextHandler::outboundReply()

Synopsis

```
virtual CORBA::Boolean CORBA::ServiceContextHandler::outboundReply  
    CORBA::Request& req,  
    CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv();
```

Description

Defines the service context handler to be called when a reply leaves the server's address space.

If not redefined in a derived class, the following implementation is inherited:

```
// C++  
{ return 1; } // no error
```

Parameters

`req` The current request that may be modified to carry a service request, or inspected to look for existing service contexts.

Return Value

- 1 (TRUE) Signals success of the Handler to Orbix.
- 0 (FALSE) Signals an error in the handler to Orbix. Orbix will display a diagnostic message in this case. The message will not be displayed if diagnostics are switched off.

CORBA::ServiceContextHandler::context_id()

- Synopsis** `virtual CORBA::ULong CORBA::ServiceContextHandler::context_id();`
- Description** Returns the context ID for the current Service Context Handler. This function does not need to be overridden by the user's service context handlers.
- Return Value** The unsigned long ID that this Service Context Handler instance should handle.

CORBA::String_var

Synopsis Class `String_var` implements the `_var` type for IDL strings required by the standard C++ mapping. The `String_var` class contains a `char*` value and ensures that this is properly freed when a `String_var` object is deallocated, for example by going out of scope.

Orbix

```
// C++
class String_var {
public:
    String_var();
    String_var(char* p);
    String_var(const char* p);

    String_var(const String_var& s);
    String_var(const String_mgr& s);
    String_var(const String_SeqElem& s);

    ~String_var();

    String_var& operator=(char* p);
    String_var& operator=(const String_var& s);
    String_var& operator=(const String_mgr& s);
    String_var& operator=(const String_SeqElem& s);

    operator ! () const;
    operator char*& ();
    operator const char* () const;

    Char& operator[] (ULong index);
    Char operator[] (ULong index) const;

    const char* in () const;
    char*& inOut ();
    char*& out ();
    char* ret () const;
};
```

Notes CORBA compliant.

CORBA::String_var::String_var()

Synopsis	<code>CORBA::String_var::String_var();</code>
Description	Default constructor.
Notes	CORBA compliant.
See Also	Other constructors.

CORBA::String_var::String_var()

Synopsis	<code>CORBA::String_var::String_var(char* p);</code>
Description	Conversion from a <code>char*</code> . The <code>String_var</code> assumes ownership of the parameter <code>p</code> .
Notes	CORBA compliant.
See Also	Other <code>String_var</code> constructors.

CORBA::String_var::String_var()

Synopsis	<code>CORBA::String_var::String_var(const CORBA::String_var& s);</code>
Description	Copy constructor.
Notes	CORBA compliant.
See Also	Other <code>String_var</code> constructors.

CORBA::String_var::~~String_var()

Synopsis	<code>CORBA::String_var::~~String_var();</code>
Description	Destructor. The destructor frees the underlying <code>char*</code> array.
Notes	CORBA compliant.

CORBA::String_var::operator=()

- Synopsis** `CORBA::String_var& CORBA::String_var::operator=(char* p);`
`CORBA::String_var& CORBA::String_var::operator=(const String_var& s);`
- Description** Assignment operators allowing assignment from a `char*` and from another `String_var`.
- Notes** CORBA compliant.

CORBA::String_var::operator[]()

- Synopsis** `char& CORBA::String_var::operator[] (CORBA::ULong index);`
`char CORBA::String_var::operator[] (CORBA::ULong index) const;`
- Description** Subscript operators to allow read and write access of the characters in the string.
- Notes** CORBA compliant.

CORBA::String_var::char*()

- Synopsis** `CORBA::String_var::operator const CORBA::String_var::char* ()`
`const;`
- Description** Converts `String_var` object to a `char*`.
- Notes** CORBA compliant.

CORBA::SystemException

Synopsis

The system exceptions are organised into a class hierarchy: each system exception is a derived class of `CORBA::SystemException` (which in turn is a derived class of `CORBA::Exception`). This allows all system exceptions to be caught in a single C++ `catch` clause.

The CORBA specification defines a set of system exceptions and Orbix adds a number of system exceptions that may be raised by Orbix to this set. The system exceptions defined by CORBA and Orbix are listed in Appendix B, “System Exceptions” on page 433.

The `IT_ERRORS` entry in the Orbix configuration file may be used to specify an alternative error messages file for system exceptions if required. Refer to the *Orbix Administrator's Guide C++ Edition* for details.

Within the errors file, you can insert comments using “//”, and you can use “\” as a continuation character if the message needs to extend past the end of line. IDL compiler errors have been divided into pre-processing, syntax and semantic errors, and their error numbers are arranged within these divisions.

Orbix

```
// C++
class SystemException : public Exception {
public:
    SystemException();
    SystemException(const SystemException&);
    virtual ~SystemException();

    const SystemException& operator=(const SystemException&);

    SystemException(ULong minor_id,
        CompletionStatus completed_status);

    CompletionStatus completed() const;

    void completed(CompletionStatus completed_status);

    void minor(ULong minor_val);

    static SystemException* _narrow(Exception* e);
```

```
        ULong minor() const;

        friend ostream& operator<<(ostream&,
            SystemException*);
    };
```

Notes The CORBA specification does not mandate a particular implementation for the `SystemException` class—the Orbix implementation described here is compliant.

See Also `CORBA::Exception`
`CORBA::UserException`
`CORBA::Environment`

CORBA::SystemException::SystemException()

Synopsis `CORBA::SystemException::SystemException();`

Description Default constructor.

Notes CORBA compliant. It should not be necessary for you to use this constructor.

See Also Other `SystemException` constructors.

CORBA::SystemException::SystemException()

Synopsis `CORBA::SystemException::SystemException(CORBA::ULong minor_id,
 CORBA::CompletionStatus completion_status);`

Description Constructs a new system exception with given minor code and `CompletionStatus`.

Notes CORBA compliant. It should not be necessary for you to use this constructor.

See Also `CORBA::Exception`
`CORBA::CompletionStatus`
`CORBA::SystemException::minor()`
Other `SystemException` constructors.

CORBA::SystemException::SystemException()

- Synopsis** `CORBA::SystemException::SystemException(
const CORBA::SystemException&);`
- Description** Copy constructor.
- Notes** CORBA compliant. It should not be necessary for you to use this constructor.
- See Also** Other `SystemException` constructors.

CORBA::SystemException::~~SystemException()

- Synopsis** `virtual CORBA::SystemException::~~SystemException();`
- Description** Destructor.
- Notes** CORBA compliant. It should not be necessary for you to use this destructor.

CORBA::SystemException::operator=()

- Synopsis** `const CORBA::SystemException& CORBA::SystemException::operator = (
const CORBA::SystemException&);`
- Description** Assignment operator.
- Notes** CORBA compliant. It should not be necessary for you to use this operator.

CORBA::SystemException::operator<<()

- Synopsis** `friend ostream& CORBA::SystemException::operator<<(ostream& o,
CORBA::SystemException* se);`
- Description** Overloads `operator<<()` to output the `SystemException` `se` on ostream `o`. The output of this operator takes the format:
`<_major>: <_id> <explanatory text string> \n`
- Notes** Orbix specific.

CORBA::SystemException::_narrow()

Synopsis	<pre>static CORBA::SystemException* CORBA::SystemException::_narrow(CORBA::Exception* e);</pre>
Description	Narrows Exception <i>e</i> to a <code>SystemException</code> . If the runtime type of <i>e</i> is not of class <code>SystemException</code> or one of its derived classes, <code>_narrow()</code> returns a null pointer. Otherwise, <code>_narrow()</code> returns a valid <code>SystemException</code> pointer. If <i>e</i> is a null pointer, <code>_narrow()</code> also returns a null pointer. Use of this function is necessary only when a C++ compiler does not support C++ exception handling.
Notes	CORBA compliant.

CORBA::SystemException::completed()

Synopsis	<pre>CORBA::CompletionStatus CORBA::SystemException::completed() const;</pre>
Description	Returns an indication of the status of an operation at the time the exception was raised. This is one of <code>COMPLETED_YES</code> , <code>COMPLETED_NO</code> , or <code>COMPLETED_MAYBE</code> .
Notes	CORBA compliant.
See Also	<code>CORBA::SystemException::completed()</code> <code>CORBA::CompletionStatus</code>

CORBA::SystemException::completed()

Synopsis	<pre>void CORBA::SystemException::completed(CORBA::CompletionStatus completion_status);</pre>
Description	Sets the status of an operation at the time an exception is raised. This is one of <code>COMPLETED_YES</code> , <code>COMPLETED_NO</code> , or <code>COMPLETED_MAYBE</code> .
Notes	CORBA compliant. It should not be necessary for you to use this function.
See Also	<code>CORBA::SystemException::completed()</code> <code>CORBA::CompletionStatus</code>

CORBA::CompletionStatus

Synopsis

```
enum CORBA::SystemException::CompletionStatus {  
    CORBA::COMPLETED_YES, CORBA::COMPLETED_NO,  
    CORBA::COMPLETED_MAYBE };
```

Description

Enumerates the possible operation completion status values at the time an exception is raised.

COMPLETED_YES The requested operation had completed processing prior to the exception being raised.

COMPLETED_NO The requested operation was never initiated.

COMPLETED_MAYBE It is indeterminate whether the requested operation was ever initiated, and if it was whether it completed processing prior to the exception being raised.

Notes

CORBA compliant.

See Also

CORBA::SystemException::completed()

CORBA::SystemException::minor()

Synopsis

```
CORBA::ULong CORBA::SystemException::minor() const;
```

Description

Returns a code describing the type of the system exception. In Orbix, this code is used to index into the `ErrorMsgs` file to extract the appropriate message.

Notes

CORBA compliant.

See Also

CORBA::minor(CORBA::ULong minor_id)

CORBA::SystemException::minor()

Synopsis

```
void CORBA::SystemException::minor(CORBA::ULong minor_id);
```

Description

Sets the code, describing the type of the system exception.

Notes

CORBA compliant. It should not be necessary for you to use this function.

See Also

CORBA::minor()

CORBA::ThreadFilter

Synopsis

`CORBA::ThreadFilter` is a derived class of class `CORBA::Filter`. A per-process filter's `inRequestPreMarshal()` function can create a thread to handle an incoming request. To do this, the filter must inherit from `CORBA::ThreadFilter`. The `inRequestPreMarshal()` function should use an underlying threads package—for example, the Solaris threads package—to create a thread, and the thread should then handle the request, possibly by instructing Orbix to send the invocation to the target object. The `inRequestPreMarshal()` function should return `-1` to Orbix to indicate that a dispatch on a new thread has occurred.

Instances of `CORBA::ThreadFilter` are installed before any user filters in the filter chain. More than one thread filter can be installed, but processing of the chain of thread filters ceases once one of them creates a thread (processing of the filter chain continues if the `inRequestPreMarshal()` function of a thread filter returns `1`). Deriving from class `CORBA::ThreadFilter` ensures that the thread filter is at the start of the filter chain.

A server has a single thread to handle requests if it has no thread filter. Thus a non-threaded server handles one request at a time. If a server makes a nested remote call, the server temporarily buffers further requests if they arrive before the remote call returns.

Orbix

```
// C++
class CORBA::ThreadFilter : public CORBA::Filter {
protected:
    ThreadFilter();
};
```

Notes

Orbix specific.

See Also

`CORBA::Filter`
`CORBA::AuthenticationFilter`

CORBA::ThreadFilter::ThreadFilter()

Synopsis

`CORBA::ThreadFilter::ThreadFilter();`

Description

The constructor adds the newly created filter object to the chain immediately before any user filters (this is the end of the per-process filter chain if there are no user filters).

Direct instances of `ThreadFilter` cannot be created: the constructor is protected to enforce this.

Notes

Orbix specific.

CORBA::TypeCode

Synopsis

The C++ class `CORBA::TypeCode` implements the IDL pseudo interface `TypeCode`. `TypeCode` is used to describe arbitrary complex IDL type structures at runtime. A `TypeCode` consists of a *kind* and a sequence of *parameters* (a parameter is of type `CORBA::Any`). The kind classifies the `TypeCode`: for example, whether it is a basic type, a struct, a sequence and so on. The parameters give the details of the type definition. For example, the IDL type `sequence<long, 20>` has the kind `tk_sequence` and has parameters `long` and `20`.

The parameters of each `TypeCode` are:

KIND	PARAMETER LIST
<code>tk_null</code>	NONE
<code>tk_void</code>	NONE
<code>tk_short</code>	NONE
<code>tk_long</code>	NONE
<code>tk_longlong</code>	NONE
<code>tk_ushort</code>	NONE
<code>tk_ulong</code>	NONE
<code>tk_ulonglong</code>	NONE
<code>tk_float</code>	NONE
<code>tk_double</code>	NONE
<code>tk_boolean</code>	NONE
<code>tk_char</code>	NONE
<code>tk_octet</code>	NONE
<code>tk_any</code>	NONE

KIND	PARAMETER LIST
tk_TypeCode	NONE
tk_Principal	NONE
tk_fixed	{ digits, scale }
tk_objref	{ interface-id }
tk_struct	{ struct-name, member-name, TypeCode, ...<repeat pairs>... }
tk_union	{ union-name, switch-TypeCode, label-value, member-name, TypeCode, ...<repeat triples>... }
tk_enum	{ enum-name, enum-id, ...<repeat enum-id>... }
tk_string	{ maxlen-integer }
tk_sequence	{ TypeCode, maxlen-integer }
tk_array	{ TypeCode, length-integer, <repeat length-integer>... }

A `TypeCode` of kind `tk_fixed` has two parameters: two integers representing the digits, and the scale of the fixed type.

A `TypeCode` of kind `tk_objref` has a single parameter giving the interface name.

A `TypeCode` of kind `tk_struct` has one parameter giving the `struct` name, and has two parameters for each member of the `struct`: the first giving the member's name and the second giving its `TypeCode`. A `struct` with N members has $2N+1$ parameters.

A `TypeCode` of kind `tk_union` has parameters giving the union name, the `TypeCode` of the switch (discriminator) of the union, and then three parameters for each member of the union:

- the label value

- the member name
- the member's `TypeCode`.

If the union has a default member, the triple for this has a label-value of 0 and the `TypeCode` of the corresponding `any` returned by `parameter()` is the `TypeCode` for an `octet`, which is not a valid switch type for a union. Thus this 0 can be distinguished from a normal 0 switch value.

A `TypeCode` of kind `tk_enum` has one parameter giving the `enum` name, and then one parameter for each enumerate constant. Enumerate constants are represented as strings.

A `TypeCode` of kind `tk_string` has one parameter—an integer giving the maximum length of the string. A 0 length indicates an unbounded string.

A `TypeCode` of kind `tk_sequence` has two parameters: a `TypeCode` for the element types, and a `CORBA::ULong` for the length. A 0 length indicates an unbounded sequence.

A `TypeCode` of kind `tk_array` has $N+1$ parameters, where N is the number of dimensions of the array. The first parameter is a `TypeCode` for the element types; the remainder are of type `long`.

Note that a `TypeCode` for an IDL exception is of kind `tk_struct` and has the same parameters as a `TypeCode` for a `struct`.

An IDL operation with a parameter of type `TypeCode` is translated into a C++ function with a parameter of type `TypeCode_ptr`. This is an object reference for a `TypeCode`. A declaration for the object which it references can be generated by the IDL compiler from named type definitions that appear in an IDL file—that is, from the following types:

```
interface
typedef
struct
union
enum
```

A number of `TypeCode` object reference constants are always available to allow the user to access `TypeCodes` for standard types. They are in `CORBA.h`:

```
CORBA::_tc_null           CORBA::_tc_void
CORBA::_tc_short         CORBA::_tc_long
CORBA::_tc_ushort        CORBA::_tc_ulong
```

CORBA::_tc_float	CORBA::_tc_double
CORBA::_tc_boolean	CORBA::_tc_char
CORBA::_tc_octet	CORBA::_tc_any
CORBA::_tc_TypeCode	CORBA::_tc_Principal
CORBA::_tc_Object	CORBA::_tc_string
CORBA::_tc_NamedValue	

CORBA

```
// IDL
// In module CORBA-ac
enum TCKind {
    tk_null, tk_void,
    tk_short, tk_long, tk_ushort, tk_ulong,
    tk_float, tk_double, tk_boolean, tk_char,
    tk_octet, tk_any, tk_TypeCode, tk_Principal,
    tk_objref, tk_struct, tk_union, tk_enum,
    tk_string, tk_sequence, tk_array,
    tk_alias, tk_except, tk_longlong, tk_ulonglong,
    tk_longdouble, tk_wchar, tk_wstring, tk_fixed
};
exception Bounds {};
pseudo interface TypeCode {
    TCKind kind();
    long param_count();
    any parameter(in long index) raises(Bounds);
    boolean equal(in TypeCode tc);
};
```

Orbix

```
// C++
class TypeCode {
public:
    TypeCode();

    TypeCode(const TypeCode&);

    ~TypeCode();

    const TypeCode& operator=(const TypeCode& src);

    operator char*() const;

    CORBA::Boolean equal(TypeCode_ptr tc,
```



```
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;
    int operator==(const TypeCode& tc) const;
    int operator!=(const TypeCode& tc) const;

    TCKind kind(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;

    Long param_count(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;

    Any parameter(Long index,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;
    static TypeCode_ptr IT_create(const char* tc,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

    static TypeCode_ptr IT_create(const TypeCode_ptr& tc,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

    static TypeCode_ptr _duplicate(
        TypeCode_ptr,
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());

    static TypeCode_ptr _nil(
        CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());
};
```

CORBA::TypeCode::TypeCode()

Synopsis	<code>CORBA::TypeCode::TypeCode();</code>
Description	Default constructor. The <code>TypeCode</code> is created with the default kind, <code>tk_null</code> .
Notes	Orbix specific.
See Also	<code>CORBA::TypeCode::IT_create()</code> Other <code>TypeCode</code> constructor.

CORBA::TypeCode::TypeCode()

Synopsis	<code>CORBA::TypeCode::TypeCode(const CORBA::TypeCode&);</code>
Description	Copy constructor.
Notes	Orbix specific.
See Also	<code>CORBA::TypeCode::IT_create()</code> Other <code>TypeCode</code> constructor.

CORBA::TypeCode::~TypeCode()

Synopsis	<code>CORBA::TypeCode::~TypeCode();</code>
Description	Destructor.
Notes	Orbix specific.

CORBA::TypeCode::operator=()

Synopsis	<code>const CORBA::TypeCode& CORBA::TypeCode::operator = (const CORBA::TypeCode& tc);</code>
Description	Assignment operator.
Notes	Orbix specific.

CORBA::TypeCode::operator==(())

Synopsis	<code>int CORBA::TypeCode::operator==(const CORBA::TypeCode& tc) const;</code>
Description	Compares self with <code>tc</code> . Two <code>TypeCodes</code> are equal when the IDL definitions from which they are compiled denote equal types.
Return Value	Returns 1 (<code>TRUE</code>) if the <code>TypeCodes</code> are equal; returns 0 (<code>FALSE</code>) otherwise.
Notes	Orbix specific. <code>CORBA::TypeCode::equal()</code> is the CORBA compliant version of this function.
See Also	<code>CORBA::TypeCode::equal()</code> <code>CORBA::TypeCode::operator!=(())</code>

CORBA::TypeCode::operator!=(0)

- Synopsis** `int CORBA::TypeCode::operator!=(const CORBA::TypeCode& tc) const;`
- Description** Compares self with `tc`. Two `TypeCodes` are equal when the IDL definitions from which they are compiled denote equal types.
- Return Value** Returns 1 (TRUE) if the `TypeCodes` are not equal; returns 0 (FALSE) otherwise.
- Notes** Orbix specific.
- See Also** `CORBA::TypeCode::operator==()`
`CORBA::TypeCode::equal()`

CORBA::TypeCode::_duplicate()

- Synopsis** `static CORBA::TypeCode_ptr CORBA::TypeCode::_duplicate(
CORBA::TypeCode_ptr obj,
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Increments the reference count of `obj`.
- Return Value** Returns the object whose reference count has been incremented.
- Notes** CORBA compliant.
- See Also** `CORBA::release()`

CORBA::TypeCode::_nil()

- Synopsis** `static CORBA::TypeCode_ptr CORBA::TypeCode::_nil(
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** Returns a nil object reference for a `TypeCode`.
- Notes** CORBA compliant.
- See Also** `CORBA::is_nil()`

CORBA::TypeCode::equal()

- Synopsis** `CORBA::Boolean CORBA::TypeCode::equal(CORBA::TypeCode_ptr tc,
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;`
- Description** Compares self with `tc`. Two `TypeCode`s are equal when the IDL definitions from which they are compiled denote equal types.
- Return Value** Returns 1 (`TRUE`) if the `TypeCode`s are equal; returns 0 (`FALSE`) otherwise.
- Notes** CORBA compliant.
- See Also** `CORBA::TypeCode::operator==()`
`CORBA::TypeCode::operator!=()`

CORBA::TypeCode::IT_create()

- Synopsis** `static CORBA::TypeCode_ptr CORBA::TypeCode::IT_create(
const CORBA::TypeCode_ptr& tc,
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv());`
- Description** In the absence of a CORBA specified way to create a `TypeCode` pseudo object in the current standard C++ mapping, Orbix provides the `IT_create()` function to initialise an object reference for a `TypeCode`.
- Use of this function is recommended in preference to C++ operator `new` to ensure memory management consistency.
- Notes** Orbix specific

CORBA::TypeCode::kind()

- Synopsis** `TCKind CORBA::TypeCode::kind(
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;`
- Description** Finds the kind of the `TypeCode`, an enumerated value of type `TCKind`.
- Notes** CORBA compliant.

CORBA::TypeCode::param_count()

- Synopsis** `CORBA::Long CORBA::TypeCode::param_count(
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;`
- Description** Finds the number of parameters belonging to the `TypeCode`. For example, the IDL type sequence<long, 20> has two parameters: long and 20.
- Return Value** Returns the number of parameters.
- Notes** CORBA compliant.

CORBA::TypeCode::parameter()

- Synopsis** `CORBA::Any CORBA::TypeCode::parameter(CORBA::Long index,
CORBA::Environment& IT_env = CORBA::IT_chooseDefaultEnv())
const;`
- Description** Finds the parameter specified by `index`. For example, the IDL type sequence<long, 20> has two parameters: long and 20. Parameters are indexed from 0 to (`param_count()` - 1).
- Exceptions** A `CORBA::Bounds` exception is raised if an attempt is made to access a non-existent parameter.
- Notes** CORBA compliant.

CORBA::UserCVHandler

Synopsis

Orbix provides a configuration file, `iona.cfg`, to configure Orbix. On UNIX, environment variables may override the entries specified in the Orbix configuration file. On Windows NT and Windows 95, Orbix is, by default, configured using the System Registry. On OpenVMS, symbols can override the entries specified in the Orbix configuration file.

You may additionally configure aspects of Orbix at runtime by providing one or more configuration value handlers that configure some or all configuration entries.

Class `UserCVHandler` is an abstract base class that defines the interface for Orbix configuration value handlers. You can create a configuration value handler by implementing a derived class of `CORBA::UserCVHandler`, creating an instance of this class and activating it.

You can arrange active configuration handlers explicitly using the functions `CORBA::Orbix::PlaceCVHandlerBefore()` and `CORBA::Orbix::PlaceCVHandlerAfter()`. If not explicitly ordered, handlers are called in reverse order of instantiation: the last handler to be instantiated is the first handler to be called.

Note: If you are migrating from Orbix 2.x and use `PlaceCVHandlerBefore()` or `PlaceCVHandlerAfter()`, you should update your code to specify `IT_ScopedConfigFile` instead of the old `IT_ConfigFile` or `IT_Registry` handlers. Refer to the *Orbix Administrator's Guide* for more details.

Orbix

```
// C++
class UserCVHandler {
public:
    UserCVHandler(const char* identifier);
    virtual ~UserCVHandler();
    virtual unsigned int GetValue(const char* name,
        char*& value) = 0;
};
```

Notes Orbix specific.

See Also CORBA::ORB::GetConfigValue()
CORBA::ORB::SetConfigValue()
CORBA::ORB::ActivateCVHandler()
CORBA::ORB::DeactivateCVHandler()
CORBA::ORB::PlaceCVHandlerBefore()
CORBA::ORB::PlaceCVHandlerAfter()
CORBA::ORB::ReinitialiseConfig()

CORBA::UserCVHandler::UserCVHandler()

Synopsis CORBA::UserCVHandler::UserCVHandler(const char* identifier);

Description Constructor.

Parameters

identifier The name of the configuration value handler.

Notes Orbix specific.

CORBA::UserCVHandler::~~UserCVHandler()

Synopsis virtual CORBA::UserCVHandler::~~UserCVHandler();

Description Destructor.

Notes Orbix specific.

CORBA::UserCVHandler::GetValue()

Synopsis virtual unsigned int CORBA::UserCVHandler::GetValue(
const char* name, char*& value) = 0;

Description Obtains the value of the configuration entry named in *name*. A derived class must implement this function.

Parameters

`name` The name of the configuration entry, for example, `IR_DAEMON_PORT`.
`value` The value of the configuration entry specified in `name`, for example, `"1507"`.

Return Value Returns `1` (`TRUE`) if a value for the specified configuration entry is found, returns `0` (`FALSE`) otherwise. If no value is found by this handler (that is, `GetValue()` returns `0`) the next configuration value handler in the list, if any, is tried.

Notes Orbix specific.

CORBA::UserException

Synopsis The user exceptions are organized into a class hierarchy: each user exception is mapped to a derived class of `CORBA::UserException` (which in turn is a derived class of `CORBA::Exception`).

Orbix

```
// C++
class UserException : public CORBA::Exception {
public:
    UserException();
    UserException(const UserException&);

    static UserException* _narrow(Exception* e);

};
```

Notes The CORBA specification does not mandate a particular implementation for the `UserException` class—the implementation described here is compliant.

See Also

- `CORBA::Exception`
- `CORBA::SystemException`
- `CORBA::Environment`

CORBA::UserException::UserException()

Synopsis `CORBA::UserException::UserException();`

Description Default constructor.

Notes CORBA compliant.

See Also Other `UserException` constructors.

CORBA::UserException::UserException()

Synopsis	<code>CORBA::UserException::UserException(const CORBA::UserException&);</code>
Description	Copy constructor.
Notes	CORBA compliant.
See Also	Other <code>UserException</code> constructors.

CORBA::UserException::operator=()

Synopsis	<code>const CORBA::UserException& CORBA::UserException::operator = (const CORBA::UserException&);</code>
Description	Assignment operator.
Notes	CORBA compliant.

CORBA::UserException::_narrow()

Synopsis	<code>static CORBA::UserException* CORBA::UserException::_narrow(CORBA::Exception* e);</code>
Description	<p>Narrows Exception <code>e</code> to a <code>UserException</code>. If the runtime type of <code>e</code> is not of class <code>UserException</code> or one of its derived classes, <code>_narrow()</code> returns a null pointer. Otherwise, <code>_narrow()</code> returns a valid <code>UserException</code> pointer. If <code>e</code> is a null pointer, <code>_narrow()</code> also returns a null pointer. Use of this function is necessary only when a C++ compiler does not support C++ exception handling.</p> <p>A version of this function is generated by the IDL compiler for each user-defined exception.</p>
Notes	CORBA compliant.

CORBA::UserOutput

Synopsis

The Orbix default output handler directs all diagnostic messages to the standard output stream, `cout`. You can provide an additional output handler by implementing a derived class of the abstract base class `CORBA::UserOutput`, creating an instance of this class and activating this instance as an output handler.

An output handler may be activated using the static function `CORBA::ORB::ActivateOutputHandler()`. More than one output handler may be active at any time. Messages are sent to all active output handlers allowing, for example, all output to be directed to standard output and logged to a file.

An output handler may be deactivated using the static function `CORBA::ORB::DeactivateOutputHandler()`.

The Orbix default output handler is named `"IT_StdOutHandler"`. It may be deactivated as follows:

```
// C++
CORBA::ORB::DeactivateOutputHandler("IT_StdOutHandler");
```

Output handlers are invoked from an application by calling one of the `Output()` functions defined on `CORBA::ORB`, for example:

```
// C++
CORBA::ORB::Output("Error: client exiting...", 2);
```

Orbix

```
// C++
class UserOutput {
public:
    UserOutput(const char* identifier);
    virtual ~UserOutput();
    virtual void Output(const char* message, int i = 1) = 0;
};
```

Notes

Orbix specific.

See Also

```
CORBA::ORB::ActivateOutputHandler()
CORBA::ORB::DeactivateOutputHandler()
CORBA::ORB::setDiagnostics()
CORBA::ORB::Output(char*, int level)
CORBA::ORB::Output(Environment&, int level)
CORBA::ORB::Output(SystemException*, int level)
```

CORBA::UserOutput::UserOutput()

Synopsis `CORBA::UserOutput::UserOutput(const char* identifier);`

Description Constructor.

Parameters

`identifier` The name of this output handler.

Notes Orbix specific.

CORBA::UserOutput::~~UserOutput()

Synopsis `virtual CORBA::UserOutput::~~UserOutput();`

Description Destructor.

Notes Orbix specific.

CORBA::UserOutput::Output()

Synopsis `virtual void CORBA::UserOutput::Output(const char* message,
int level = 1) = 0;`

Description A derived class may redefine this function to direct diagnostic messages to a location other than the default `cout` stream.

Parameters

`message` The message to be output.

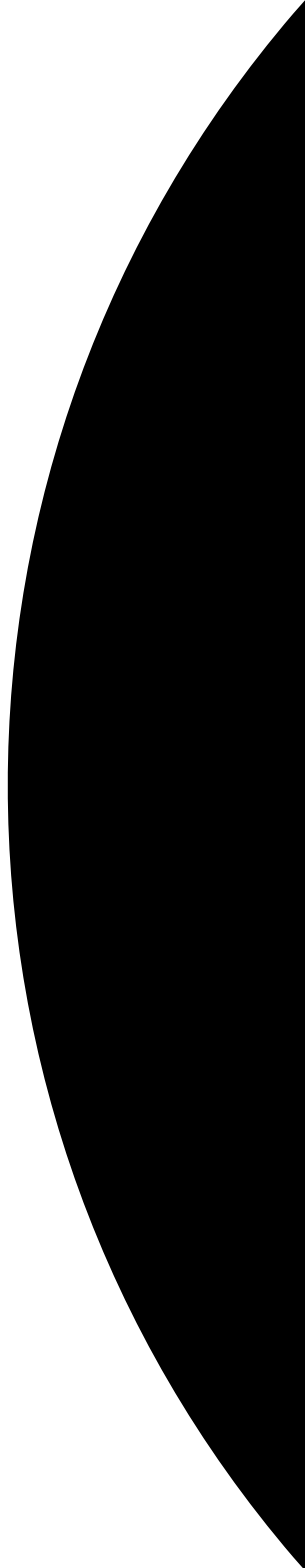
`level` The diagnostic level for this message. The default value for Orbix diagnostic messages is level 1.)

Notes Orbix specific.

See Also `CORBA::ORB::setDiagnostics()`

Part II

IDL Interface to the Interface Repository



Common CORBA Data Types

CORBA::DefinitionKind

Synopsis

```
enum DefinitionKind {  
    dk_none, dk_all,  
    dk_Attribute, dk_Constant, dk_Exception, dk_Interface,  
    dk_Module, dk_Operation, dk_Typedef,  
    dk_Alias, dk_Struct, dk_Union, dk_Enum,  
    dk_Primitive, dk_String, dk_Sequence, dk_Array,  
    dk_Repository};
```

Description

Each IFR object has an attribute (`def_kind`) of type `DefinitionKind` that records the kind of the IFR object. For example, the `def_kind` attribute of an `InterfaceDef` object are `dk_interface`. The enumeration constants `dk_none` and `dk_all` have special meanings when searching for an object in a repository.

Notes

CORBA compliant.

CORBA::Identifier

Synopsis

```
typedef string Identifier;
```

Description

A simple name that identifies modules, interfaces, constants, typedefs, exceptions, attributes, and operations. An identifier is not necessarily unique within the entire Interface Repository; it is unique only within a particular `Repository`, `ModuleDef`, `InterfaceDef`, or `OperationDef`.

Notes

CORBA compliant.

CORBA::RepositoryId

- Synopsis** `typedef string RepositoryId;`
- Description** A string that uniquely identifies a module, interface, constant, typedef, exception, attribute, or operation.
- Notes** CORBA compliant.

CORBA::ScopedName

- Synopsis** `typedef string ScopedName;`
- Description** A `ScopedName` gives an entity's name relative to a scope. A `ScopedName` that begins with “:.” is an *absolute scoped name*; one that uniquely identifies an entity within a repository. An example might be:
- ```
 ::Account::makeWithdrawal.
```
- A `ScopedName` that does not begin with “:.” is a relative scoped name; one that identifies an entity relative to some other entity. For example:

```
 makeWithdrawal
```

This is within the entity with the absolute scoped name `::Account`.

**Notes** CORBA compliant.

# CORBA::AliasDef

**Synopsis** The interface `AliasDef` describes an IDL typedef that aliases another definition. It is used to represent an IDL typedef.

```
CORBA // IDL
// In module CORBA.

interface AliasDef : TypedefDef {
 attribute IDLType original_type_def;
};
```

**Notes** CORBA compliant.

**See Also** `CORBA::Contained`  
`CORBA::Container::create_alias()`

## **AliasDef::describe()**

**Synopsis** `Description describe();`

**Description** Inherited from `Contained`, the `describe()` operation returns a structure of type `Contained::Description`:

```
struct Description {
 DefinitionKind kind;
 any value;
};
```

The `DefinitionKind` for the `kind` member is `dk_Alias`. The `value` member is an any whose `TypeCode` is `_tc_AliasDescription` and whose value is a structure of type `TypeDescription`:

```
// IDL
struct TypeDescription {
 Identifier name;
 RepositoryId id;
 RepositoryId defined_in;
 VersionSpec version;
 TypeCode type;
};
```

**See Also** `CORBA::TypedefDef::describe()`

### **AliasDef::original\_type\_def**

**Synopsis** `attribute IDLType original_type_def;`

**Description** Identifies the type being aliased.

Modifying the `original_type_def` attribute automatically updates the `type` attribute (the `type` attribute is inherited from `TypedefDef` which in turn inherits it from `IDLType`). Both attributes contain the same information.

**Notes** CORBA compliant.

**See Also** `CORBA::IDLType::type`

# CORBA::ArrayDef

**Synopsis** The interface `ArrayDef` represents a one-dimensional array. A multi-dimensional array is represented by an `ArrayDef` with an element type that is another array definition. The final element type represents the type of element contained in the array. You can create an instance of interface `ArrayDef` using `CORBA::Repository::create_array()`.

**CORBA**

```
// IDL
// In module CORBA

interface ArrayDef : IDLType {
 attribute unsigned long length;
 readonly attribute TypeCode element_type;
 attribute IDLType element_type_def;
};
```

**Notes** CORBA compliant.

**See Also** `CORBA::IDLType`  
`CORBA::ArrayDef::element_type_def`  
`CORBA::Repository::create_array()`

## **ArrayDef::element\_type**

**Synopsis** `readonly attribute TypeCode element_type;`

**Description** Identifies the type of the element contained in the array. This contains the same information as in the attribute `element_type_def`.

**Notes** CORBA compliant.

**See Also** `CORBA::ArrayDef::element_type_def`

### **ArrayDef::element\_type\_def**

- Synopsis**      `attribute IDLType element_type_def;`
- Description**    Describes the type of the element contained within the array. This contains the same information as in the attribute `element_type_def`.
- The type of elements contained in the array can be changed by changing this attribute. Changing this attribute also changes the `element_type` attribute.
- Notes**          CORBA compliant.
- See Also**        `CORBA::ArrayDef::element_type`

### **ArrayDef::length**

- Synopsis**      `attribute unsigned long length;`
- Description**    Specifies the number of elements in the array.
- Notes**          CORBA compliant.

# CORBA::AttributeDef

**Synopsis** Interface AttributeDef describes an IDL attribute.

**CORBA**

```
// IDL
// In module CORBA.
enum AttributeMode { ATTR_NORMAL, ATTR_READONLY };

interface AttributeDef : Contained {
 readonly attribute TypeCode type;
 attribute IDLType type_def;
 attribute AttributeMode mode;
};
```

**Notes** CORBA compliant.

**See Also** CORBA::Contained  
CORBA::InterfaceDef::create\_attribute()

## AttributeDef::describe()

**Synopsis** Description describe();

**Description** Inherited from Contained, the describe() operation returns a structure of type Contained::Description:

```
struct Description {
 DefinitionKind kind;
 any value;
};
```

The DefinitionKind for the kind member is dk\_Attribute. The value member is an any whose TypeCode is \_tc\_AttributeDescription.

The value is a structure of type `AttributeDescription`:

```
// IDL
// In module CORBA.
struct AttributeDescription {
 Identifier name;
 RepositoryId id;
 RepositoryId defined_in;
 VersionSpec version;
 TypeCode type;
 AttributeMode mode;
};
```

**See Also** `CORBA::Contained::describe()`

### **AttributeDef::mode**

**Synopsis** `attribute AttributeMode mode;`

**Description** Specifies whether the attribute is read/write (`ATTR_NORMAL`) or read-only (`ATTR_READONLY`).

**Notes** CORBA compliant.

### **AttributeDef::type**

**Synopsis** `readonly attribute TypeCode type;`

**Description** Identifies the type of this attribute. The same information is contained in the `type_def` attribute.

**Notes** CORBA compliant.

**See Also** `CORBA::TypeCode`  
`CORBA::AttributeDef::type_def`



**AttributeDef::type\_def**

|                    |                                                                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <code>attribute IDLType type_def;</code>                                                                                                                                                                         |
| <b>Description</b> | Describes the type for this attribute. The same information is contained in the <code>type</code> attribute. Changing the <code>type_def</code> attribute automatically changes the <code>type</code> attribute. |
| <b>Notes</b>       | CORBA compliant.                                                                                                                                                                                                 |
| <b>See Also</b>    | <code>CORBA::IDLType</code><br><code>CORBA::AttributeDef::type</code>                                                                                                                                            |



# CORBA::ConstantDef

**Synopsis** Interface ConstantDef describes an IDL constant. The name of the constant is inherited from Contained.

**CORBA**

```
// IDL
// in module CORBA.

interface ConstantDef : Contained {
 readonly attribute TypeCode type;
 attribute IDLType type_def;
 attribute any value;
};
```

**Notes** CORBA compliant.

**See Also** CORBA::Contained  
CORBA::Container::create\_constant()

## ConstantDef::describe()

**Synopsis** Description describe();

**Description** Inherited from Contained, the describe() operation returns a structure of type Contained::Description:

```
struct Description {
 DefinitionKind kind;
 any value;
};
```

The DefinitionKind for the kind member is dk\_Constant.

The value member is an any whose `TypeCode` is `_tc_ConstantDescription` and whose value is a structure of type `ConstantDescription`:

```
// IDL
struct ConstantDescription {
 Identifier name;
 RepositoryId id;
 RepositoryId defined_in;
 VersionSpec version;
 TypeCode type;
 any value;
};
```

**Notes** CORBA compliant.

**See Also** `CORBA::Contained::describe()`

### **ConstantDef::type**

**Synopsis** `readonly attribute TypeCode type;`

**Description** Identifies the type of this constant. The type must be a `TypeCode` for one of the simple types (such as `long`, `short`, `float`, `char`, `string`, `double`, `boolean`, `unsigned long`, and `unsigned short`). The same information is contained in the `type_def` attribute.

**Notes** CORBA compliant.

**See Also** `CORBA::ConstantDef::type_def`

### **ConstantDef::type\_def**

**Synopsis** `attribute IDLType type_def;`

**Description** Identifies the type of the constant. The same information is contained in the `type` attribute.

The type of a constant can be changed by changing its `type_def` attribute. This also changes its `type` attribute.

**Notes** CORBA compliant.

**See Also** `CORBA::ConstantDef::type`

### **ConstantDef::value**

- Synopsis**      `attribute any value;`
- Description**      Contains the value for this constant. When changing the `value` attribute, the `TypeCode` of the `any` must be the same as the `type` attribute.
- Notes**      CORBA compliant.
- See Also**      `CORBA::TypeCode`



# CORBA::Contained

## Synopsis

Interface `Contained` is an abstract interface describing Interface Repository objects that can be contained in a module, interface, or repository. It is a base interface for the following interfaces:

```
ModuleDef
InterfaceDef
ConstantDef
TypedefDef
ExceptionDef
AttributeDef
OperationDef
StructDef
EnumDef
UnionDef
AliasDef
```

## CORBA

```
// IDL
// In module CORBA.

typedef string VersionSpec;

interface Contained : IObject {
 attribute RepositoryId id;
 attribute Identifier name;
 attribute VersionSpec version;

 readonly attribute Container defined_in;
 readonly attribute ScopedName absolute_name;
 readonly attribute Repository containing_repository;

 struct Description {
 DefinitionKind kind;
 any value;
 };
 Description describe();
```

```
void move (
 in Container new_container,
 in Identifier new_name,
 in VersionSpec new_version);
};
```

**Notes** CORBA compliant.

**See Also** CORBA::Container  
CORBA::IObject

### **Contained::absolute\_name()**

**Synopsis** readonly attribute ScopedName absolute\_name;

**Description** Gives the absolute scoped name of an object.

**Notes** CORBA compliant.

**See Also** CORBA::ScopedName

### **Contained::containing\_repository()**

**Synopsis** readonly attribute Repository containing\_repository;

**Description** Gives the Repository within which the object is contained.

**Notes** CORBA compliant.

### **Contained::defined\_in**

**Synopsis** attribute Container defined\_in;

**Description** Specifies the Repository ID for the Interface Repository object in which the object is contained.

An IFR object is said to be contained by the IFR object in which it is defined. For example, an `InterfaceDef` object is contained by the `ModuleDef` in which it is defined.



---

A second notion of `Contained` applies to objects of type `AttributeDef` or `OperationDef`. These objects may also be said to be contained in an `InterfaceDef` object if they are inherited into that interface.

---

**Note:** Inheritance of operations and attributes across the boundaries of different modules is also allowed.

---

**Notes** CORBA compliant.

**See Also** `CORBA::Container::contents()`

### **Contained::describe()**

**Synopsis** `Description describe();`

**Description** Returns a structure of type `Contained::Description`:

```
// IDL
struct Description {
 DefinitionKind kind
 any value;
};
```

This is a generic form of description that is used as a wrapper for another structure stored in the `value` field. Depending on the type of the `Contained` object, the `value` field contains a corresponding description structure:

```
ConstantDescription
ExceptionDescription
AttributeDescription
OperationDescription
ModuleDescription
InterfaceDescription
TypeDescription
```

The last of these, `TypeDescription` is used for objects of type `StructDef`, `UnionDef`, `EnumDef`, and `AliasDef` (it is associated with interface `TypeDefDef` from which these four listed interfaces inherit).

The `kind` field contains the same value as the `def_kind` attribute that `Contained` inherits from `IRObject`.

**Notes** CORBA compliant.

**See Also**      `CORBA::Container::describe_contents()`  
                 `CORBA::DefinitionKind`

### **Contained::id**

**Synopsis**      `attribute RepositoryId id;`

**Description**    A `RepositoryId` provides an alternative method of naming an object which is independent of the `ScopedName`. In order to be CORBA compliant the naming conventions specified for CORBA `RepositoryIds` should be followed. Changing the `id` attribute changes the global identity of the contained object. It is an error to change the `id` to a value that currently exists in the contained object's `Repository`.

**Notes**        CORBA compliant.

### **Contained::move ()**

**Synopsis**      `void move(in Container new_container,`  
                 `in Identifier new_name, in VersionSpec new_version);`

**Description**    Removes this object from its container, and adds it to the container specified by `new_container`. The new container must:

- ♦ Be in the same repository.
- ♦ Be capable of containing an object of this type.
- ♦ Not contain an object of the same name (unless multiple versions are supported).

The `name` attribute of the object being moved is changed to that specified by the `new_name` parameter. The `version` attribute is changed to that specified by the `new_version` parameter.

**Notes**        CORBA compliant.

**See Also**      `CORBA::Container`

**Contained::name**

**Synopsis**      attribute Identifier name;

**Description**      The name of the object within its scope. For example, in the following definition:

```
// IDL
interface Example {
 void op();
};
```

the names are `Example` and `op`. A `name` must be unique within its scope but is not necessarily unique within an Interface Repository. You can change the `name` attribute but it is an error to change it to a value that is currently in use within the object's Container.

**Notes**              CORBA compliant.

**See Also**          Contained::id

**Contained::version**

**Synopsis**      attribute VersionSpec version;

**Description**      The version number for this object. Each interface object is identified by a version which distinguishes it from other versioned objects of the same name.

**Notes**              CORBA compliant.



# CORBA::Container

**Synopsis**      Interface `Container` describes objects that can contain other objects. Such objects are:

```
Repository
ModuleDef
InterfaceDef
```

**CORBA**

```
// IDL
// In module CORBA.

typedef sequence <Contained> ContainerSeq;

interface Container : IObject {

 Contained lookup(in ScopedName search_name);

 ContainedSeq contents(
 in DefinitionKind limit_type,
 in boolean exclude_inherited);

 ContainedSeq lookup_name(
 in Identifier search_name,
 in long levels_to_search,
 in DefinitionKind limit_type,
 in boolean exclude_inherited);

 struct Description {
 Contained contained_object;
 DefinitionKind kind;
 any value;
 };

 typedef sequence<Description> DescriptionSeq;

 DescriptionSeq describe_contents(
 in DefinitionKind limit_type,
 in boolean exclude_inherited,
 in long max_returned_objs);
```

```
ModuleDef create_module(
 in RepositoryId id,
 in Identifier name,
 in VersionSpec version
);
ConstantDef create_constant(
 in RepositoryId id,

 in Identifier name,
 in VersionSpec version,
 in IDLType type,
 in any value);

StructDef create_struct(
 in RepositoryId id,
 in Identifier name,
 in VersionSpec version,
 in StructMemberSeq members);

UnionDef create_union(
 in RepositoryId id,
 in Identifier name,
 in VersionSpec version,
 in IDLType discriminator_type,
 in UnionMemberSeq members);

EnumDef create_enum(
 in RepositoryId id,
 in Identifier name,
 in VersionSpec version,
 in EnumMemberSeq members);

AliasDef create_alias(
 in RepositoryId id,
 in Identifier name,
 in VersionSpec version,
 in IDLType original_type);

InterfaceDef create_interface(
 in RepositoryId id,
 in Identifier name,
 in VersionSpec version,
 in InterfaceDefSeq base_interfaces);
```

```
ExceptionDef create_exception(
 in RepositoryId id;
 in Identifier name,
 in VersionSpec version,
 in StructMemberSeq members);
};
```

**Notes** CORBA compliant.

**See Also** CORBA::IObject

### **Container::contents()**

**Synopsis** ContainedSeq contents(in DefinitionKind limit\_type,  
 in boolean exclude\_inherited);

**Description** Returns a sequence of Contained objects that are directly contained in (defined in or inherited into) the target object. You can use this operation to navigate through the hierarchy of definitions—starting, for example, at a Repository.

#### **Parameters**

|                   |                                                                                                                                                                                                                                                             |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| limit_type        | If set to dk_all, all contained Interface Repository objects are returned. If set to the DefinitionKind for a specific interface type, it returns only interfaces of that type. For example, if set to, dk_Operation, it returns contained operations only. |
| exclude_inherited | Applies only to interfaces. If set to TRUE, inherited objects are not returned. If set to FALSE, objects are returned even if they are inherited.                                                                                                           |

**Notes** CORBA compliant.

**See Also** CORBA::Container::describe\_contents()  
CORBA::DefinitionKind

### Container::create\_alias()

#### Synopsis

```
UnionDef create_alias(in RepositoryId id,
 in Identifier name,
 in VersionSpec version,
 in IDLType original_type);
```

#### Description

Creates a new `AliasDef` object within the target `Container`. The `defined_in` attribute is set to the target `Container`. The `containing_repository` attribute is set to the `Repository` in which the new `AliasDef` object is defined.

#### Parameters

|                            |                                                                                                                                                                                                                                         |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>id</code>            | The <code>Repository</code> ID for the new <code>AliasDef</code> object. An error is returned if an <code>Interface Repository</code> object with the same <code>id</code> already exists within the object's <code>Repository</code> . |
| <code>name</code>          | The name for the new <code>AliasDef</code> object. You cannot specify a name that already exists within the object's <code>Container</code> when multiple versions are not supported.                                                   |
| <code>version</code>       | A version for the new <code>AliasDef</code> .                                                                                                                                                                                           |
| <code>original_type</code> | The original type that is being aliased.                                                                                                                                                                                                |

#### Notes

CORBA compliant.

#### See Also

`CORBA::AliasDef`

### Container::create\_constant()

#### Synopsis

```
ConstantDef create_constant(in RepositoryId id,
 in Identifier name,
 in VersionSpec version,
 in IDLType type,
 in any value);
```

#### Description

Creates a `ConstantDef` object within the target `Container`. The `defined_in` attribute is set to the target `Container`. The `containing_repository` attribute is set to the `Repository` in which the new `ConstantDef` object is defined.



**Parameters**

|                      |                                                                                                                                                                                         |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>id</code>      | The Repository ID of the new <code>ConstantDef</code> object. You cannot specify an <code>id</code> that already exists within the object's Repository.                                 |
| <code>name</code>    | The name of the new <code>ConstantDef</code> object. You cannot specify a name that already exists within the object's <code>Container</code> when multiple versions are not supported. |
| <code>version</code> | The version number of the new <code>ConstantDef</code> object.                                                                                                                          |
| <code>type</code>    | The type of the defined constant. This must be one of the simple types (long, short, ulong, ushort, float, double, char, string, boolean).                                              |
| <code>value</code>   | The value of the defined constant.                                                                                                                                                      |

**Notes** CORBA compliant.

**See Also** `CORBA::ConstantDef`

**Container::create\_enum()****Synopsis**

```
EnumDef create_enum(in RepositoryId id,
 in Identifier name,
 in VersionSpec version,
 in EnumMemberSeq members);
```

**Description**

Creates a new `EnumDef` object within the target `Container`. The `defined_in` attribute is set to `Container`. The `containing_repository` attribute is set to the `Repository` in which the new `EnumDef` object is defined.

**Parameters**

|                      |                                                                                                                                                                                 |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>id</code>      | The Repository ID of the new <code>EnumDef</code> object. You cannot specify an <code>id</code> that already exists within the Repository.                                      |
| <code>name</code>    | The name of the <code>EnumDef</code> object. You cannot specify a name that already exists within the object's <code>Container</code> when multiple versions are not supported. |
| <code>version</code> | The version number of the new <code>EnumDef</code> object.                                                                                                                      |
| <code>members</code> | A sequence of <code>EnumMember</code> structures that describe each member of the new <code>EnumDef</code> object.                                                              |

**Notes** CORBA compliant.

**See Also** CORBA::EnumDef

### Container::create\_exception()

**Synopsis**

```
ExceptionDef create_exception(in RepositoryId id,
 in Identifier name,
 in VersionSpec version,
 in StructMemberSeq members);
```

**Description** Creates a new `ExceptionDef` object within the target `Container`. The `defined_in` attribute is set to `Container`. The `containing_repository` attribute is set to the `Repository` in which new `ExceptionDef` object is defined. The `type` attribute of the `StructMember` structures is ignored and should be set to `_tc_void`.

#### Parameters

|                      |                                                                                                                                                                                          |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>id</code>      | The <code>Repository</code> ID of the new <code>ExceptionDef</code> object. You cannot specify an <code>id</code> that already exists within the object's <code>Repository</code> .      |
| <code>name</code>    | The name of the new <code>ExceptionDef</code> object. You cannot specify a name that already exists within the object's <code>Container</code> when multiple versions are not supported. |
| <code>version</code> | A version number for the new <code>ExceptionDef</code> object.                                                                                                                           |
| <code>members</code> | A sequence of <code>StructMember</code> structures that describe each member of the new <code>ExceptionDef</code> object.                                                                |

**Notes** CORBA compliant.

**See Also** CORBA::ExceptionDef

**Container::create\_interface()****Synopsis**

```
InterfaceDef create_interface(
 in RepositoryId id,
 in Identifier name,
 in VersionSpec version,
 in InterfaceDefSeq base_interfaces);
```

**Description**

Creates a new empty `InterfaceDef` object within the target `Container`. The `defined_in` attribute is set to `Container`. The `containing_repository` attribute is set to the `Repository` in which the new `InterfaceDef` object is defined.

**Parameters**

|                              |                                                                                                                                                                                          |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>id</code>              | The <code>Repository ID</code> of the new <code>InterfaceDef</code> object. You cannot specify an <code>id</code> that already exists within the object's <code>Repository</code> .      |
| <code>name</code>            | The name of the new <code>InterfaceDef</code> object. You cannot specify a name that already exists within the object's <code>Container</code> when multiple versions are not supported. |
| <code>version</code>         | A version for the new <code>InterfaceDef</code> object.                                                                                                                                  |
| <code>base_interfaces</code> | A sequence of <code>InterfaceDef</code> objects from which the new interface inherits.                                                                                                   |

**Notes**

CORBA compliant.

**See Also**

`CORBA::InterfaceDef`

**Container::create\_module()****Synopsis**

```
ModuleDef create_module(
 in RepositoryId id,
 in Identifier name,
 in VersionSpec version);
```

**Description**

Creates an empty `ModuleDef` object within the target `Container`. The `defined_in` attribute is set to `Container`. The `containing_repository` attribute is set to the `Repository` in which the newly created `ModuleDef` object is defined.

### Parameters

- `id` The Repository ID of the new `ModuleDef` object. You cannot specify an `id` that already exists within the object's Repository.
- `name` The name of the new `ModuleDef` object. You cannot specify a name that already exists within the object's `Container` when multiple versions are not supported.
- `version` A version for the `ModuleDef` object to be created.

**Notes** CORBA compliant.

### **Container::create\_struct()**

#### Synopsis

```
StructDef create_struct(
 in RepositoryId id,
 in Identifier name,
 in VersionSpec version,
 in StructMemberSeq members);
```

#### Description

Creates a new `StructDef` object within the target `Container`. The `defined_in` attribute is set to `Container`. The `containing_repository` attribute is set to the Repository in which the new `StructDef` object is defined. The `type` attribute of the `StructMember` structures is ignored and should be set to `_tc_void`.

#### Parameters

- `id` The Repository ID of the new `StructDef` object. You cannot specify an `id` that already exists within the object's Repository.
- `name` The name of the new `StructDef` object. You cannot specify a name that already exists within the object's `Container` when multiple versions are not supported.
- `version` A version for the new `StructDef` object.
- `members` A sequence of `StructMember` structures that describe each member of the new `StructDef` object.

**Notes** CORBA compliant.

**See Also** `CORBA::StructDef`

---

**Container::create\_union()****Synopsis**

```
UnionDef create_union(
 in RepositoryId id,
 in Identifier name,
 in VersionSpec version,
 in IDLType discriminator_type,
 in UnionMemberSeq members);
```

**Description**

Creates a new `UnionDef` object within the target `Container`. The `defined_in` attribute is set to the target `Container`. The `containing_repository` attribute is set to the `Repository` in which the new `UnionDef` object is defined. The `type` attribute of the `UnionMember` structures is ignored and should be set to `_tc_void`.

**Parameters**

|                                 |                                                                                                                                                                                      |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>id</code>                 | The <code>Repository</code> ID of the new <code>UnionDef</code> object. You cannot specify an <code>id</code> that already exists within the object's <code>Repository</code> .      |
| <code>name</code>               | The name of the new <code>UnionDef</code> object. You cannot specify a name that already exists within the object's <code>Container</code> when multiple versions are not supported. |
| <code>version</code>            | A version for the new <code>UnionDef</code> object.                                                                                                                                  |
| <code>discriminator_type</code> | The type of the <code>Union</code> discriminator.                                                                                                                                    |
| <code>members</code>            | A sequence of <code>UnionMember</code> structures that describe each member of the new <code>UnionDef</code> object.                                                                 |

**Notes**

CORBA compliant.

**See Also**

CORBA::UnionDef

### Container::describe\_contents()

#### Synopsis

```
DescriptionSeq describe_contents(
 in DefinitionKind limit_type,
 in boolean exclude_inherited,
 in long max_returned_objs);
```

#### Description

A combination of the operation `Contained::describe()` and the operation `Container::contents()`, the `describe_contents()` operation returns a sequence of structures of type `Container::Description`:

```
// IDL
struct Description {
 Contained contained_object;
 DefinitionKind kind;
 any value;
};
```

Each of these structures gives the object reference of a contained object, together with its `kind` and `value`.

#### Parameters

#### Notes

CORBA compliant.

#### See Also

`CORBA::Container::contents()`  
`CORBA::Contained::describe()`

### Container::lookup()

#### Synopsis

```
Contained lookup(in ScopedName search_name);
```

#### Description

Locates an object name within the target container. The objects can be directly or indirectly defined in or inherited into the target container.

#### Parameters

`search_name` The name of the object to search for relative to the target container. If a relative name is given, the object is looked up relative to the target container. If `search_name` is an absolute scoped name (prefixed by `::`), the object is located relative to the containing `Repository`.

#### Notes

CORBA compliant

**See Also** CORBA::Container::lookup\_name()  
CORBA::ScopedName

### Container::lookup\_name()

**Synopsis**

```
ContainedSeq lookup_name(
 in Identifier search_name,
 in long levels_to_search,
 in DefinitionKind limit_type,
 in boolean exclude_inherited);
```

**Description** Locates an object or objects by name within the target container. The named objects can be directly or indirectly defined in or inherited into the target container.

#### Parameters

|                   |                                                                                                                                                                                                                                                                                          |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| search_name       | The simple name of the object to search for.<br>The use of the wildcard character "*" is also allowed (Orbix specific).                                                                                                                                                                  |
| levels_to_search  | Defines whether the search is confined to the current object or should include all Interface Repository objects contained by the object. If set to -1, the current object and all contained Interface Repository objects are searched. If set to 1, only the current object is searched. |
| limit_type        | If this is set to dk_all, all the contained Interface Repository objects are returned. If set to the DefinitionKind for a particular Interface Repository kind, it returns only objects of that kind. For example, if set to dk_Operation, it returns contained operations only.         |
| exclude_inherited | Applies only to interfaces. If set to TRUE, inherited objects are not returned. If set to FALSE, objects are returned even if they are inherited.                                                                                                                                        |

**Return Value** Returns a sequence of contained objects. (More than one object, having the same simple name can exist within a nested scope structure.)

**Notes** CORBA compliant.

**See Also**      `CORBA::DefinitionKind`



# CORBA::EnumDef

**Synopsis** Interface EnumDef describes an IDL enumeration definition.

**CORBA**

```
// IDL
// In module CORBA.

typedef sequence <Identifier> EnumMemberSeq;

interface EnumDef : TypedefDef {
 attribute EnumMemberSeq members;
};
```

**Notes** CORBA compliant.

**See Also** CORBA::TypedefDef

## EnumDef::describe()

**Synopsis** Description describe();

**Description** Inherited from Contained, the describe() operation returns a structure of type Contained::Description:

```
struct Description {
 DefinitionKind kind;
 any value;
};
```

The DefinitionKind for the kind member is dk\_Enum. The value member is an any whose TypeCode is \_tc\_TypeDescription and whose value is a structure of type TypeDescription:

```
// IDL
struct TypeDescription {
 Identifier name;
 RepositoryId id;
 RepositoryId defined_in;
 VersionSpec version;
 TypeCode type;
};
```

The `type` field of the `struct` gives the `TypeCode` of the defined `Enum`.

**See Also** `CORBA::TypedefDef::describe()`

### **EnumDef::members**

**Synopsis** `attribute EnumMemberSeq members;`

**Description** Contains the enumeration's list of identifiers (its enumerated constants).  
The set of enumerated constants can be changed by changing this attribute.

**Notes** CORBA compliant.

**See Also** `CORBA::Identifier`

# CORBA::ExceptionDef

**Synopsis** Interface ExceptionDef describes an IDL exception. It inherits from interface Contained.

**CORBA**

```
// IDL
// In module CORBA.
struct StructMember {
 Identifier name;
 TypeCode type;
 IDLType type_def;
};
typedef sequence <StructMember> StructMemberSeq;

interface ExceptionDef : Contained {
 readonly attribute TypeCode type;
 attribute StructMemberSeq members;
};
```

**Notes** CORBA compliant.

**See Also** CORBA::Contained

## ExceptionDef::describe()

**Synopsis** Description describe();

**Description** Inherited from Contained, the describe() operation returns a structure of type Contained::Description:

```
struct Description {
 DefinitionKind kind;
 any value;
};
```

The `DefinitionKind` for the `kind` member is `dk_Exception`. The `value` member is an any whose `TypeCode` is `_tc_ExceptionDescription` and whose value is a structure of type `ExceptionDescription`:

```
// IDL
struct ExceptionDescription {
 Identifier name;
 RepositoryId id;
 RepositoryId defined_in;
 VersionSpec version;
 TypeCode type;
};
```

The `type` field of the struct gives the `TypeCode` of the defined exception.

**Notes**

CORBA compliant.

**See Also**

`CORBA::Contained::describe()`  
`CORBA::TypeCode`

### **ExceptionDef::members**

**Synopsis**

`attribute StructMemberSeq members;`

**Description**

In a sequence of `StructMember` structures, the `members` attribute describes the exception's members.

You can modify the `members` attribute to change the structure's members. You should set only the `name` and `type_def` fields of each `StructMember` (set the `type` field to `_tc_void`, and it is set automatically to the `TypeCode` of the `type_def` field).

**Notes**

CORBA compliant.

**See Also**

`CORBA::StructDef`  
`CORBA::ExceptionDef::type`

## ExceptionDef::type

- Synopsis**      readonly attribute TypeCode type;
- Description**      The type of the exception (from which you can understand the definition of the exception). The `TypeCode` kind for an exception is `tk_except`.
- Notes**              CORBA compliant.
- See Also**            `CORBA::TypeCode`  
`CORBA::ExceptionDef::members`



# CORBA::IDLType

**Synopsis** The abstract interface `IDLType` describes Interface Repository objects that represent interfaces, type definitions, structures, unions, enumerations, aliases (that is, IDL typedef), primitives, bounded strings, sequences, and array types. Thus, it is a base interface for the following interfaces:

```
ArrayDef
InterfaceDef
PrimitiveDef
StringDef
SequenceDef
TypedefDef
AliasDef
StructDef
UnionDef
EnumDef
```

**CORBA**

```
// IDL
// In module CORBA.
interface IDLType : IRObject {
 readonly attribute TypeCode type;
};
```

**Notes** CORBA compliant.

**See Also** `CORBA::IRObject`  
`CORBA::TypeCode`

## IDLType::type

**Synopsis** `readonly attribute TypeCode type;`

**Description** Encodes the type information of an Interface Repository object. You can also extract most type information using operations and attributes defined on derived types of `IDLType`.

**Notes** CORBA compliant.

**See Also** `CORBA::TypeCode`





# CORBA::IObject

**Synopsis** The interface `IObject` provides a base interface from which all interface repository interfaces are derived.

**CORBA**

```
// IDL
// In module CORBA.

interface IObject {
 readonly attribute DefinitionKind def_kind;
 void destroy ();
};
```

**Notes** CORBA compliant.

**See Also** `CORBA::DefinitionKind`

## **IObject::def\_kind**

**Synopsis** `readonly attribute DefinitionKind def_kind`

**Description** Identifies the kind of an IFR object. For example, an `OperationDef` object, describing an IDL operation, has the kind `dk_Operation`.

**Notes** CORBA compliant.

**See Also** `CORBA::DefinitionKind`

## **IObject::destroy()**

**Synopsis** `void destroy();`

**Description** Deletes an IFR object. This also deletes any objects contained within the target object. You cannot invoke the `destroy()` operation on a `Repository` or on a `PrimitiveDef` object.

**Notes** CORBA compliant.



# CORBA::IT\_Repository

**Synopsis** The interface `IT_Repository` provides transactional access to the Interface Repository. These operations can be used to help ensure that the repository is left in a consistent state. In the present implementation only one transaction may be active at a time.

**CORBA**

```
// IDL
// In module CORBA.

interface IT_Repository : Repository {
 unsigned long start();
 void commit(in unsigned long transaction_id);
 void rollBack(in unsigned long transaction_id);
 typedef sequence <unsigned long> transactions;
 transactions active_transactions();
};
```

**Notes** Orbix specific.

**See Also** `CORBA::Container`  
`CORBA::Repository`

## **IT\_Repository::start()**

**Synopsis** `unsigned long start();`

**Description** Starts a new transaction.

**Return Value** Returns a transaction identifier for the transaction started or 0 if the transaction could not be started.

**Notes** Orbix specific.

### **IT\_Repository::commit()**

**Synopsis** `void commit(in unsigned long transaction_id);`

**Description** Commits a transaction.

**Parameters**

`transaction_id` The identifier for the transaction.

**Notes** Orbix specific.

**See Also** `CORBA::IT_Repository::start()`

### **IT\_Repository::rollBack()**

**Synopsis** `void rollBack(in unsigned long transaction_id);`

**Description** Rolls back all changes made during the transaction to the previous commit point.

**Parameters**

`transaction_id` The identifier for the transaction to be rolled back.

**Notes** Orbix specific.

**See Also** `CORBA::IT_Repository::commit()`

### **IT\_Repository::active\_transactions()**

**Synopsis** `transactions active_transactions();`

**Description** Returns a sequence of active `transaction_ids`. If no transaction is active, an empty sequence is returned.

**Notes** Orbix specific.

# CORBA::ModuleDef

**Synopsis** The interface `ModuleDef` describes an IDL module. It inherits from the interfaces `Container` and `Contained`.

**CORBA**

```
// IDL
// In module CORBA.
interface ModuleDef : Container, Contained {
};
```

**Notes** CORBA compliant.

**See Also** `CORBA::Contained`  
`CORBA::Container`

## ModuleDef::describe()

**Synopsis** `Description describe();`

**Description** Inherited from `Contained`, the `describe()` operation returns a structure of type `Contained::Description`:

```
struct Description {
 DefinitionKind kind;
 any value;
};
```

The `DefinitionKind` for the `kind` member is `dk_Module`. The `value` member is an any whose `TypeCode` is `_tc_ModuleDescription` and whose value is a structure of type `ModuleDescription`:

```
struct ModuleDescription {
 Identifier name;
 RepositoryId id;
 RepositoryId defined_in;
 VersionSpec version;
};
```

**Notes** CORBA compliant.

**See Also** `CORBA::Contained::describe()`



# CORBA::OperationDef

**Synopsis** Interface `OperationDef` describes an IDL operation that is defined in an IDL interface.

One use of `OperationDef` is to construct an `NVList` for a specific operation for use in the Dynamic Invocation Interface. See

`CORBA::ORB::create_operation_list()` for details.

## CORBA

```
// IDL
// In module CORBA.

enum OperationMode { OP_NORMAL, OP_ONEWAY };

enum ParameterMode { PARAM_IN, PARAM_OUT, PARAM_INOUT };

struct ParameterDescription {
 Identifier name;
 TypeCode type;
 IDLType type_def;
 ParameterMode mode;
};

typedef sequence <ParameterDescription> ParDescriptionSeq;

typedef Identifier ContextIdentifier;
typedef sequence <ContextIdentifier> ContextIdSeq;

typedef sequence <ExceptionDescription> ExcDescriptionSeq;

interface OperationDef : Contained {
 readonly attribute TypeCode result;
 attribute IDLType result_def;
 attribute ParDescriptionSeq params;
 attribute OperationMode mode;
 attribute ContextIdSeq contexts;
 attribute ExceptionDefSeq exceptions;
};
```

**Notes** CORBA compliant.

**See Also**      CORBA::Contained  
                 CORBA::ORB::create\_operation\_list()  
                 CORBA::ExceptionDef

### **OperationDef::contexts**

**Synopsis**      attribute ContextIdSeq contexts;  
**Description**    The list of context identifiers specified in the context clause of the operation.  
**Notes**         CORBA compliant.

### **OperationDef::exceptions**

**Synopsis**      attribute ExceptionDefSeq  
**Description**    The list of exceptions that the operation can raise.  
**Notes**         CORBA compliant.  
**See Also**      CORBA::ExceptionDef

### **OperationDef::describe()**

**Synopsis**      Description describe();  
**Description**    Inherited from Contained, the describe() operation returns a structure of type Contained::Description:

```
struct Description {
 DefinitionKind kind;
 any value;
};
```

The DefinitionKind for the kind member is dk\_Operation. The value member is an any whose TypeCode is \_tc\_OperationDescription and whose value is a structure of type OperationDescription:



```
struct OperationDescription {
 Identifier name;
 RepositoryId id;
 RepositoryId defined_in;
 VersionSpec version;
 TypeCode result;
 OperationMode mode;
 ContextIdSeq contexts;
 ParDescriptionSeq parameters;
 ExcDescriptionSeq exceptions;
};
```

**Notes** CORBA compliant.

**See Also** CORBA::Contained::describe()  
CORBA::ExceptionDef

### OperationDef::mode

**Synopsis** attribute OperationMode mode;

**Description** Specifies whether the operation is normal (OP\_NORMAL) or oneway (OP\_ONEWAY). The mode attribute can only be set to OP\_ONEWAY if the result is `_tc_void` and all parameters have a mode of PARAM\_IN.

**Notes** CORBA compliant.

### OperationDef::params

**Synopsis** attribute ParDescriptionSeq params;

**Description** Specifies the parameters for this operation. It is a sequence of structures of type ParameterDescription:

```
struct ParameterDescription {
 Identifier name;
 TypeCode type;
 IDLType type_def;
 ParameterMode mode;
};
```

The `name` member provides the name for the parameter. The `type` member identifies the `TypeCode` for the parameter. The `type_def` member identifies the definition of the type for the parameter. The `mode` specifies whether the parameter is an `in` (`PARAM_IN`), an `out` (`PARAM_OUT`) or an `inout` (`PARAM_INOUT`) parameter. The order of the `ParameterDescriptions` is significant.

**Notes** CORBA compliant.

**See Also** `CORBA::TypeCode`  
`CORBA::IDLType`

### **OperationDef::result**

**Synopsis** `readonly attribute TypeCode result;`

**Description** The return type of this operation. The attribute `result_def` contains the same information.

**Notes** CORBA compliant.

**See Also** `CORBA::TypeCode`  
`CORBA::OperationDef::result_def`

### **OperationDef::result\_def**

**Synopsis** `attribute IDLType result_def;`

**Description** Describes the return type for this operation. The attribute `result_def` contains the same information.

Setting the `result_def` attribute also updates the `result` attribute.

**Notes** CORBA compliant.

**See Also** `CORBA::IDLType`  
`CORBA::OperationDef::result`

# CORBA::PrimitiveDef

**Synopsis** Interface `PrimitiveDef` represents a primitive type such as short or long. `PrimitiveDef` objects are anonymous (unnamed) and owned by the Repository. Objects of type `PrimitiveDef` cannot be created. When needed, you can obtain a reference to a `PrimitiveDef` through a call to the operation `CORBA::Repository::get_primitive()`.

```
// IDL
// In module CORBA.

enum PrimitiveKind {
 pk_null, pk_void, pk_short, pk_long, pk_ushort, pk_ulong,
 pk_float, pk_double, pk_boolean, pk_char, pk_octet,
 pk_any, pk_TypeCode, pk_Principal, pk_string, pk_objref
};

interface PrimitiveDef : IDLType {
 readonly attribute PrimitiveKind kind;
};
```

**Notes** CORBA compliant.

**See Also** `CORBA::IDLType`

## **PrimitiveDef::kind**

**Synopsis** `readonly attribute PrimitiveKind kind;`

**Description** Identifies which of the primitive types is represented by this `PrimitiveDef`. A `PrimitiveDef` with a kind of type `pk_string` represents an unbounded string, a bounded string is represented by the interface `StringDef`. A `PrimitiveDef` with a kind of type `pk_objref` represents the IDL type `Object`.

**Notes** CORBA compliant.

**See Also** `CORBA::IDLType`  
`CORBA::Object`  
`CORBA::StringDef`



# CORBA::Repository

**Synopsis** The Interface Repository itself is a container for IDL type definitions. Its interface is described by the `Repository` interface. You can use it to look up any definition, by either name or identity, that is defined in the global name space or within an interface or module.

```
CORBA // IDL
// In module CORBA.

interface Repository : Container {
 Contained lookup_id(
 in RepositoryId search_id);

 PrimitiveDef get_primitive (in PrimitiveKind kind);

 StringDef create_string (in unsigned long bound);

 SequenceDef create_sequence (
 in unsigned long bound,
 in IDLType element_type);

 ArrayDef create_array (
 in unsigned long length,
 in IDLType element_type);
};
```

**Notes** CORBA compliant.

**See Also** `CORBA::Container`

### Repository::create\_array()

**Synopsis**      `ArrayDef create_array(in unsigned long length,  
                          in IDLType element_type);`

**Description**      Returns a new array object defining an anonymous (unnamed) type. The new array object must be used in the definition of exactly one other object; it is deleted when the object it is contained in is deleted. It is the application's responsibility to delete any anonymous type object it creates if subsequently that object is not successfully used in the definition of a `Contained` object.

**Parameters**

`length`            The number of elements in the array.  
    `element_type`    The type of element that the array contains.

**Notes**            CORBA compliant.

**See Also**        `CORBA::ArrayDef`  
                  `CORBA::IObject`

### Repository::create\_sequence()

**Synopsis**      `SequenceDef create_sequence (in unsigned long bound,  
                              in IDLType element_type);`

**Description**      Returns a new sequence object defining an anonymous (unnamed) type. The new sequence object must be used in the definition of exactly one other object; it is deleted when the object it is contained in is deleted. It is the application's responsibility to delete any anonymous type object it creates if subsequently that object is not successfully used in the definition of a `Contained` object.

**Parameters**

`bound`             The number of elements in the sequence. A bound of 0 indicates an unbounded sequence.  
    `element_type`    The type of element that the sequence contains.

**Notes**            CORBA compliant.

**See Also**        `CORBA::SequenceDef`

### Repository::create\_string()

- Synopsis** `StringDef create_string (in unsigned long bound);`
- Description** Returns a new string object defining an anonymous (unnamed) type. The new string object must be used in the definition of exactly one other object; it is deleted when the object it is contained in is deleted. It is the application's responsibility to delete any anonymous type object it creates if subsequently that object is not successfully used in the definition of a `Contained` object.
- Parameters**
- `bound` The maximum number of characters in the string. This cannot be 0.
- Notes** CORBA compliant.
- See Also** `CORBA::StringDef`

### Repository::get\_primitive()

- Synopsis** `PrimitiveDef get_primitive(in PrimitiveKind kind);`
- Description** Returns a reference to a `PrimitiveDef` of the specified `PrimitiveKind`. All `PrimitiveDefs` are owned by the `Repository`, one primitive object per primitive type (for example, `short`, `long`, `unsigned short`, `unsigned long` and so on).
- Notes** CORBA compliant.
- See Also** `CORBA::PrimitiveDef`

### Repository::describe\_contents()

- Synopsis** `sequence<Description> describe_contents (`  
`in InterfaceName restrict_type,`  
`in boolean exclude_inherited,`  
`in long max_returned_objs);`
- Description** The operation `describe_contents()` is inherited from interface `Container`. It returns a sequence of `Container::Description` structures; one such structure for each top level item in the repository. The structure is defined as:

```
// IDL
```

```
struct Description {
 Contained contained_object;
 DefinitionKind kind;
 any value;
};
```

Each structure has the following members:

|                  |                                                                                                                                                                                                                                                                                                                                                          |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| contained_object | The object reference, of type <code>Contained</code> , of the contained top-level object. You can call the <code>describe()</code> function on an object reference, of type <code>Contained</code> , to get further information on a top level object in the Repository.                                                                                 |
| kind             | The kind of the object being described.                                                                                                                                                                                                                                                                                                                  |
| value            | An any that may contain one of the following structs:<br><br><code>ModuleDescription</code><br><code>ConstantDescription</code><br><code>TypeDescription</code><br><code>ExceptionDescription</code><br><code>AttributeDescription</code><br><code>ParameterDescription</code><br><code>OperationDescription</code><br><code>InterfaceDescription</code> |

**Notes** CORBA compliant.

**See Also** `Container::describe_contents()`  
`DefinitionKind`

### **Repository::lookup\_id()**

**Synopsis** `Contained lookup_id(in RepositoryId search_id);`

**Description** Returns an object contained within the Repository given its `RepositoryId`.

**Notes** CORBA compliant.

**See Also** `Contained`



# CORBA::SequenceDef

**Synopsis** Interface SequenceDef represents an IDL sequence definition. It inherits from the interface IDLType.

**CORBA**

```
// IDL
// In module CORBA.
interface SequenceDef : IDLType {
 attribute unsigned long bound;
 readonly attribute TypeCode element_type;
 attribute IDLType element_type_def;
};
```

**Notes** CORBA compliant.

**See Also** CORBA::IDLType  
CORBA::Repository::create\_sequence()

## SequenceDef::bound

**Synopsis** attribute unsigned long bound;

**Description** The bound of the sequence. A bound of 0 indicates an unbounded sequence type. Changing the bound attribute also updates the inherited type attribute.

**Notes** CORBA compliant.

**See Also** CORBA::SequenceDef::type

### SequenceDef::element\_type

- Synopsis**      `readonly attribute TypeCode element_type;`
- Description**    Describes the type of the element contained within this sequence. The attribute `element_type` contains the same information.
- Notes**          CORBA compliant.
- See Also**        `CORBA::element_type_def`

### SequenceDef::element\_type\_def

- Synopsis**      `attribute IDLType element_type_def;`
- Description**    Describes the type of element contained within this sequence. The attribute `element_type_def` contains the same information. Setting the `element_type_def` attribute also updates the `element_type` and `IDLType::type` attributes.
- Notes**          CORBA compliant.
- See Also**        `CORBA::SequenceDef::element_type`  
`CORBA::IDLType::type`

### SequenceDef::type

- Synopsis**      `readonly attribute TypeCode type;`
- Description**    The `type` attribute is inherited from interface `IDLType`. This attribute is a `tk_sequence TypeCode` that describes the sequence. It is updated automatically whenever the attributes `bound` or `element_type_def` are changed.
- Notes**          CORBA compliant.
- See Also**        `CORBA::SequenceDef::element_type_def`  
`CORBA::SequenceDef::bound`

# CORBA::StringDef

**Synopsis** Interface `StringDef` represents a bounded string type. Unbounded strings are primitive types. A `StringDef` object is anonymous, that is, unnamed.

**CORBA**

```
// IDL
// In module CORBA.
interface StringDef : IDLType {
 attribute unsigned long bound;
};
```

**Notes** CORBA compliant.

**See Also** `CORBA::IDLType`  
`CORBA::PrimitiveDef`  
`CORBA::Repository::create_string()`

## **StringDef::bound**

**Synopsis** `attribute unsigned long bound;`

**Description** Specifies the bound of the string. This cannot be zero.

**Notes** CORBA compliant.



# CORBA::StructDef

**Synopsis** Interface StructDef describes an IDL structure.

```
CORBA // IDL
// In module CORBA.
struct StructMember {
 Identifier name;
 TypeCode type;
 IDLType type_def;
};

typedef sequence<StructMember> StructMemberSeq;

interface StructDef : TypedefDef {
 attribute StructMemberSeq members;
};
```

**Notes** CORBA compliant.

**See Also** CORBA::Contained  
CORBA::Container::create\_struct()

## StructDef::describe()

**Synopsis** Description describe();

**Description** Inherited from Contained, the describe() operation returns a structure of type Contained::Description:

```
struct Description {
 DefinitionKind kind;
 any value;
};
```

The `DefinitionKind` for the `kind` member is `dk_Struct`. The `value` member is an any whose `TypeCode` is `_tc_TypeDescription` and whose value is a structure of type `TypeDescription`:

```
// IDL
// In module CORBA.
struct TypeDescription {
 Identifier name;
 RepositoryId id;
 RepositoryId defined_in;
 VersionSpec version;
 TypeCode type;
};
```

**Notes** CORBA compliant.

**See Also** `CORBA::TypedefDef::describe()`

### **StructDef::members**

**Synopsis** `attribute StructMemberSeq members;`

**Description** Describes the members of the structure.

You can modify this attribute to change the members of a structure. Only the `name` and `type_def` fields of each `StructMember` should be set (the `type` field should be set to `_tc_void` and it is set automatically to the `TypeCode` of the `type_def` field).

**Notes** CORBA compliant.

**See Also** `CORBA::TypedefDef`

# CORBA::TypedefDef

**Synopsis** The abstract interface `TypedefDef` is inherited by all IFR interfaces (except `InterfaceDef`) that define named types. Named types are types for which a name must appear in their definition such as structures, unions, enumerations and aliases.

Anonymous types (`PrimitiveDef`, `StringDef`, `SequenceDef` and `ArrayDef`) do not inherit from `TypedefDef`.

The role of interface `TypedefDef` in the IFR is not of particular importance; it is merely a base interface for `StructDef`, `UnionDef`, `EnumDef` and `AliasDef`.

**CORBA**

```
// IDL
// In module CORBA.

interface TypedefDef : Contained, IDLType {
};
```

**Notes** CORBA compliant.

**See Also** `CORBA::Contained`

## TypedefDef::describe()

**Synopsis** `Description describe();`

**Description** Inherited from `Contained`, the `describe()` operation returns a structure of type `Contained::Description`:

```
struct Description {
 DefinitionKind kind;
 any value;
};
```

The `DefinitionKind` for the `kind` member is `dk_Typedef`. The value member is an any whose `TypeCode` is `_tc_TypeDescription` and whose value is a structure of type `TypeDescription`:

```
// IDL
// In module CORBA.
struct TypeDescription {
 Identifier name;
 RepositoryId id;
 RepositoryId defined_in;
 VersionSpec version;
 TypeCode type;
};
```

**See Also** `CORBA::Contained::describe()`



# CORBA::UnionDef

**Synopsis** Interface UnionDef represents an IDL union.

```
CORBA // IDL
// In module CORBA.

struct UnionMember {
 Identifier name;
 any label;
 TypeCode type;
 IDLType type_def;
};

typedef sequence <UnionMember> UnionMemberSeq;

interface UnionDef : TypedefDef {
 readonly attribute TypeCode discriminator_type;
 attribute IDLType discriminator_type_def;
 attribute UnionMemberSeq members;
};
```

**Notes** CORBA compliant.

**See Also** CORBA::Contained  
CORBA::TypedefDef  
CORBA::Container::create\_union()

## UnionDef::describe()

**Synopsis** Description describe();

**Description** Inherited from Contained, the describe() operation returns a structure of type Contained::Description:

```
struct Description {
 DefinitionKind kind;
 any value;
};
```

The `DefinitionKind` for the `kind` member is `dk_Union`. The `value` member is an any whose `TypeCode` is `_tc_TypeDescription` and whose `value` is a structure of type `TypeDescription`:

```
// IDL
struct TypeDescription {
 Identifier name;
 RepositoryId id;
 RepositoryId defined_in;
 VersionSpec version;
 TypeCode type;
};
```

**See Also** `CORBA::TypedefDef::describe()`

### **UnionDef::discriminator\_type**

**Synopsis** `readonly attribute TypeCode discriminator_type;`

**Description** Describes the discriminator type for this union. For example, if the union currently contains a `long`, the `discriminator_type` is `_tc_long`. The attribute `discriminator_type_def` contains the same information.

**Notes** CORBA compliant.

**See Also** `CORBA::TypeCode`

### **UnionDef::discriminator\_type\_def**

**Synopsis** `attribute IDLType discriminator_type_def;`

**Description** Describes the discriminator type for this union. The attribute `discriminator_type` contains the same information.

Changing this attribute automatically updates the `discriminator_type` attribute and the `IDLType::type` attribute.

**Notes** CORBA compliant.

**See Also** `CORBA::IDLType::type`  
`CORBA::UnionDef::dis`

**UnionDef::members****Synopsis**

attribute UnionMemberSeq members;

**Description**

Contains a description of each union member: its name, label and type (`type` and `type_def` contain the same information).

You can modify the `members` attribute to change the union's members. You should set only the `name`, `label` and `type_def` fields of each `UnionMember` (set the `type` field to `_tc_void`, and it is set automatically to the `TypeCode` of the `type_def` field).

**Notes**

CORBA compliant.

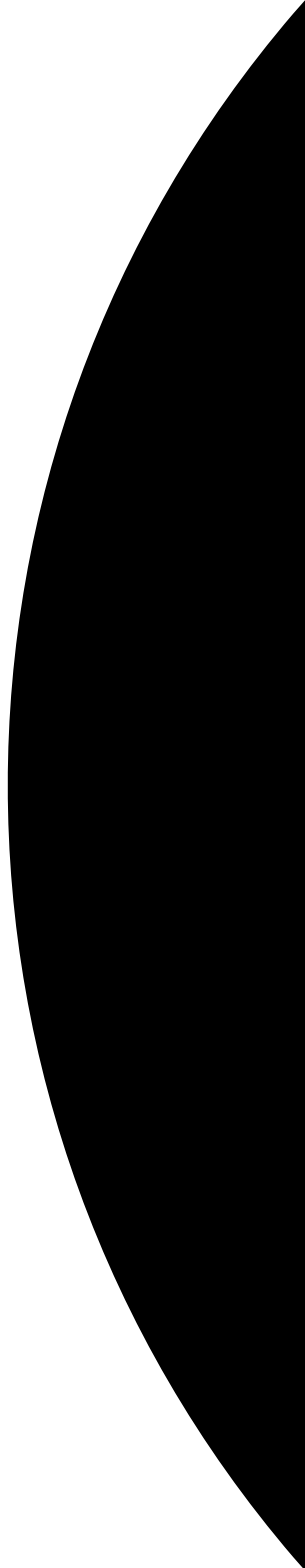
**See Also**

CORBA::TypedefDef



## Part III

# IDL Interface to the Orbix Daemon





# IDL Interface to the Orbix Daemon

## Synopsis

The Orbix daemon is an Orbix server with an IDL interface called `IT_daemon`. The Orbix daemon is responsible for launching servers (if an appropriate server is not already running), and dispatching operation requests. The daemon is only involved, if at all, with the first operation request from a client—it is not involved with subsequent requests. The Orbix daemon executable is called `orbixd`.

The Orbix daemon is also responsible for managing the Implementation Repository. It accepts all requests from the Orbix utilities—`putit`, `catit`, `lsit` and so on—and from Orbix Server Manager.

As an Orbix server, applications may bind to the Orbix daemon in a similar way as for other servers. For example, you can use the following code to bind to the Orbix daemon on host `targetHost`:

```
// C++
#include <daemon.hh>

IT_daemon_var daemon;
try {
 daemon = IT_daemon::_bind("0:IT_daemon", targetHost);
} ...
```

## Orbix

```
// IDL
interface IT_daemon{
 boolean lookUp(in string service,
 out stringSeq hostList,
 in octet hops,
 in string tag);

 boolean addHostsToServer(in string server,
 in stringSeq hostList);
 boolean addHostsToGroup(in string group,
 in stringSeq hostList);
 boolean addGroupsToServer(in string server,
 in stringSeq groupList);
 boolean delHostsFromServer(in string server,
 in stringSeq hostList);
 boolean delHostsFromGroup(in string group,
```

```
 in stringSeq hostList);
boolean delGroupsFromServer(in string server,
 in stringSeq groupList);
boolean listHostsInServer(in string server,
 out stringSeq hostList);
boolean listHostsInGroup(in string group,
 out stringSeq hostList);
boolean listGroupsInServer(in string server,
 out stringSeq groupList);

enum LaunchStatus {
 inActive,
 manualLaunch,
 automaticLaunch
};

struct serverDetails {
 string server;
 string marker;
 string principal;
 string code;
 string comms;
 string port;
 unsigned long OSspecific;
 LaunchStatus status;
};

void listActiveServers(out serverDetailsSeq servers);

void killServer(in string name, in string marker);

void newSharedServer(in string serverName,
 in stringSeq marker,
 in stringSeq launchCommand,
 in unsigned long mode_flags);
void newUnSharedServer(in string serverName,
 in stringSeq marker,
 in stringSeq launchCommand,
 in unsigned long mode_flags);

void newPerMethodServer(in string serverName,
 in stringSeq method,
 in stringSeq launchCommand);
```



## IDL Interface to the Orbix Daemon

---

```
void listServers(in string subdir,
 out stringSeq servers);

void deleteServer(in string serverName);

boolean serverExists(in string serverName);

void getServer(in string serverName,
 out string commsProtocol,
 out string codeProtocol,
 out string activationPolicy,
 out unsigned long mode_flags,
 out string owner,
 out string invokeList,
 out string launchList,
 out stringSeq markers,
 out stringSeq methods,
 out stringSeq commands);

void addUnsharedMarker(in string serverName,
 in string markerName,
 in string newCommand);

void removeUnsharedMarker(in string serverName,
 in string markerName);

void addSharedMarker(in string serverName,
 in string markerName,
 in string newCommand);

void removeSharedMarker(in string serverName,
 in string markerName);

void addMethod(in string serverName,
 in string methodName,
 in string newCommand);

void removeMethod(in string serverName,
 in string methodName);

void newDirectory(in string dirName);
void deleteDirectory(in string dirName,
 in boolean deleteChildren);

void changeOwnerServer(in string new_owner,
 in string serverName);

void addInvokeRights(in string userGroup,
 in string serverName);

void removeInvokeRights(in string userGroup,
 in string serverName);
```

```
void addLaunchRights(in string userGroup,
 in string serverName);
void removeLaunchRights(in string userGroup,
 in string serverName);
void addInvokeRightsDir(in string userGroup,
 in string dirName);

void removeInvokeRightsDir(in string userGroup,
 in string dirName);
void addLaunchRightsDir(in string userGroup,
 in string dirName);
void removeLaunchRightsDir(in string userGroup,
 in string dirName);
};
```

**Notes** Orbix specific.

### **IT\_daemon::addDirRights()**

**Synopsis** void IT\_daemon::addDirRights(in string userGroup, in string dirName);

**Description** Adds the user or group in userGroup to the list of owners for the directory dirName.

**Notes** Orbix specific.

### **IT\_daemon::addInvokeRights()**

**Synopsis** void IT\_daemon::addInvokeRights(  
in string userGroup,  
in string serverName);

**Description** Adds the user or group in userGroup to the *invoke* access control list (ACL) for the server serverName. A user who has invoke rights on a server can invoke operations on any object controlled by that server. By default, only the owner of an Implementation Repository entry has invoke rights on the server registered.

**Notes** Orbix specific.

**See Also** IT\_daemon::RemoveInvokeRights()  
IT\_daemon::addLaunchRights()

### IT\_daemon::addInvokeRightsDir()

**Synopsis**

```
void IT_daemon::addInvokeRightsDir(
 in string userGroup,
 in string dirName);
```

**Description**

Adds the user or group in *userGroup* to the *invoke* access control list (ACL) for the directory *dirName*.

**Notes**

Orbix specific.

**See Also**

```
IT_daemon::RemoveInvokeRightsDir()
IT_daemon::AddInvokeRights()
```

### IT\_daemon::addLaunchRights()

**Synopsis**

```
void IT_daemon::addLaunchRights(
 in string userGroup,
 in string serverName);
```

**Description**

Adds the user or group in *userGroup* to the *launch* access control list for the server *serverName*. By default, only the owner of an Implementation Repository entry has launch rights on the server registered.

**Notes**

Orbix specific.

**See Also**

```
IT_daemon::removeLaunchRights()
```

### IT\_daemon::addLaunchRightsDir()

**Synopsis**

```
void IT_daemon::addLaunchRightsDir(
 in string userGroup,
 in string dirName);
```

**Description**

Adds the user or group in *userGroup* to the *launch* access control list for the directory *dirName*.

**Notes**

Orbix specific.

**See Also**

```
IT_daemon::addLaunchRights()
IT_daemon::removeLaunchRightsDir()
```

### IT\_daemon::addMethod()

**Synopsis**

```
void IT_daemon::addMethod(
 in string serverName,
 in string methodName,
 in string newCommand);
```

**Description**

Adds an *activation order* to the Implementation Repository entry for the per-method server, `serverName`. This activation order specifies that an invocation of a method whose name matches the method (or method pattern) indicated in `methodName` should cause the server to be launched using the command `newCommand`.

**Notes**

Orbix specific.

**See Also**

`IT_daemon::removeMethod()`

### IT\_daemon::addSharedMarker()

**Synopsis**

```
void IT_daemon::addSharedMarker(
 in string serverName,
 in string markerName,
 in string newCommand);
```

**Description**

Adds an *activation order* to the Implementation Repository entry for the shared server, `serverName`. This activation order specifies that an invocation for an object whose marker matches the marker (or marker pattern) indicated in `markerName` should cause the server to be launched (if not already running) using the command, `newCommand`.

**Notes**

Orbix specific.

**See Also**

`IT_daemon::removeSharedMarker()`

### IT\_daemon::addUnsharedMarker()

**Synopsis**

```
void IT_daemon::addUnsharedMarker(
 in string serverName,
 in string markerName,
 in string newCommand);
```

- Description** Adds an *activation order* to the Implementation Repository entry for the unshared server, `serverName`. This activation order specifies that an invocation for an object whose marker matches the marker (or marker pattern) indicated in `markerName` should cause the server to be launched using the command, `newCommand`.
- Notes** Orbix specific.
- See Also** `IT_daemon::removeUnsharedMarker()`

### **IT\_daemon::changeOwnerServer()**

- Synopsis**  

```
void IT_daemon::changeOwnerServer(
 in string new_owner,
 in string serverName);
```
- Description** Changes the ownership of the Implementation Repository entry for the server `serverName`. The user invoking this operation must be the current owner of the Implementation Repository entry.
- Notes** Orbix specific.

### **IT\_daemon::deleteDirectory()**

- Synopsis**  

```
void IT_daemon::deleteDirectory(
 in string dirName,
 in boolean deleteChildren);
```
- Description** Removes a registration directory from the Implementation Repository. If `deleteChildren` is true, server entries and sub-directories are also deleted.
- Notes** Orbix specific.
- See Also** `IT_daemon::newDirectory()`

### **IT\_daemon::deleteServer()**

- Synopsis**  

```
void IT_daemon::deleteServer(
 in string serverName);
```

**Description** Deletes the entry for the server, `serverName`, from the Implementation Repository.

**Notes** Orbix specific.

**See Also** `IT_daemon::newPerMethodServer()`  
`IT_daemon::newSharedServer()`  
`IT_daemon::newUnsharedServer()`

### **IT\_daemon::getServer()**

**Synopsis**

```
void IT_daemon::getServer(
 in string serverName,
 out string commsProtocol,
 out string codeProtocol,
 out string activationPolicy,
 out unsigned long mode_flags,
 out string owner,
 out string invokeList,
 out string launchList,
 out stringSeq markers,
 out stringSeq methods,
 out stringSeq commands);
```

**Description** Gets full information about the Implementation Repository entry for `serverName`.

**Notes** Orbix specific.

### **IT\_daemon::killServer()**

**Synopsis**

```
void IT_daemon::killServer(
 in string name,
 in string marker);
```

**Description** Kills a server process. Where there is more than one server process, the marker parameter is used to select between different processes. The marker parameter is required when killing a process in the unshared mode.

**Notes** Orbix specific.

### **IT\_daemon::LaunchStatus**

- Synopsis** `enum IT_daemon::LaunchStatus {  
 inActive,  
 manualLaunch,  
 automaticLaunch  
};`
- Description** Possible values for the `LaunchStatus` of a server.
- Notes** Orbix specific.

### **IT\_daemon::listActiveServers()**

- Synopsis** `typedef sequence<serverDetails> serverDetailsSeq;  
void IT_daemon::listActiveServers(  
 out serverDetailsSeq servers);`
- Description** Returns a list of active server processes known to the Orbix daemon and includes information about each process.
- Notes** Orbix specific.
- See Also** `IT_daemon::serverDetails`

### **IT\_daemon::listServers()**

- Synopsis** `void IT_daemon::listServers(  
 in string subdir,  
 out stringSeq servers);`
- Description** Lists all servers in the Implementation Repository directory `subdir`.
- Notes** Orbix specific.

### IT\_daemon::newDirectory()

- Synopsis** `void IT_daemon::newDirectory(in string dirName);`
- Description** Creates a new Implementation Repository directory. The name is specified in `dirName` and may be a new directory or a subdirectory of an existing directory. The '/' character is used to indicate a subdirectory—for example, the name "server/banks" is a valid directory name.
- Notes** Orbix specific.
- See Also** `IT_daemon::deleteDirectory()`

### IT\_daemon::newPerMethodServer()

- Synopsis** `void IT_daemon::newPerMethodServer(  
    in string serverName,  
    in stringSeq methods,  
    in stringSeq launchCommands);`
- Description** Creates a new entry in the Implementation Repository for the per-method server `serverName`. The new entry has an activation order for each element of the sequences in `methods` and `launchCommands`.
- Parameters**
- |                             |                                                                                                                                                                           |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>serverName</code>     | The name of the server.                                                                                                                                                   |
| <code>methods</code>        | A sequence of methods. Each element in the sequence has a corresponding element in the sequence <code>launchCommands</code> .                                             |
| <code>launchCommands</code> | A sequence of launch commands (the full path name of an executable file). Each element in the sequence has a corresponding element in the sequence <code>methods</code> . |
- Notes** Orbix specific.



### IT\_daemon::newSharedServer()

#### Synopsis

```
void IT_daemon::newSharedServer(
 in string serverName,
 in stringSeq markers,
 in stringSeq launchCommands,
 in unsigned long mode_flags);
```

#### Description

Creates a new Implementation Repository entry for the shared server `serverName`. The new entry has an activation order for each element of the sequences in `markers` and `launchCommands`.

#### Parameters

|                             |                                                                                                                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>serverName</code>     | The name of the server.                                                                                                                                                                                      |
| <code>markers</code>        | A sequence of markers. Each element in the sequence has a corresponding element in the sequence <code>launchCommands</code> .                                                                                |
| <code>launchCommands</code> | A sequence of launch commands (the full path name of an executable file and possibly command line switches). Each element in the sequence has a corresponding element in the sequence <code>markers</code> . |
| <code>mode_flags</code>     | Further activation mode details.<br><ul style="list-style-type: none"><li>0 Multiple-client activation mode.</li><li>1 Per-client activation mode.</li><li>2 Per-client-process activation mode.</li></ul>   |

#### Notes

Orbix specific.

### IT\_daemon::newUnSharedServer()

#### Synopsis

```
void IT_daemon::newUnSharedServer(
 in string serverName,
 in stringSeq markers,
 in stringSeq launchCommands,
 in unsigned long mode_flags);
```

#### Description

Creates a new Implementation Repository entry for the unshared server `serverName`. The new entry has an activation order for each element of the sequences in `markers` and `launchCommands`.

#### Parameters

|                             |                                                                                                                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>serverName</code>     | The name of the server.                                                                                                                                                                                      |
| <code>markers</code>        | A sequence of markers. Each element in the sequence has a corresponding element in the sequence <code>launchCommands</code> .                                                                                |
| <code>launchCommands</code> | A sequence of launch commands (the full path name of an executable file and possibly command line switches). Each element in the sequence has a corresponding element in the sequence <code>markers</code> . |
| <code>mode_flags</code>     | Further activation mode details.<br><ul style="list-style-type: none"><li>0 Multiple-client activation mode.</li><li>1 Per-client activation mode.</li><li>2 Per-client-process activation mode.</li></ul>   |

#### Notes

Orbix specific.

### IT\_daemon::removeDirRights()

#### Synopsis

```
void IT_daemon::removeDirRights(in string userGroup, in string
 dirName);
```

#### Description

Removes the user or group in `userGroup` to the list of owners for the directory `dirName`.

#### Notes

Orbix specific.

### **IT\_daemon::removeInvokeRights()**

- Synopsis**      `void IT_daemon::removeInvokeRights (  
                  in string userGroup,  
                  in string serverName);`
- Description**    Removes the user or group in `userGroup` from the `invoke` access control list for server `serverName`.
- Notes**          Orbix specific.
- See Also**        `IT_daemon::AddInvokeRights()`

### **IT\_daemon::removeInvokeRightsDir()**

- Synopsis**      `void IT_daemon::removeInvokeRightsDir (  
                  in string userGroup,  
                  in string dirName);`
- Description**    Removes the user or group in `userGroup` from the *invoke* access control list for directory `dirName`.
- Notes**          Orbix specific.
- See Also**        `IT_daemon::AddInvokeRightsDir()`

### **IT\_daemon::removeLaunchRights()**

- Synopsis**      `void IT_daemon::removeLaunchRights (  
                  in string userGroup,  
                  in string serverName);`
- Description**    Removes the user or group in `userGroup` from the *launch* access control list for server `serverName`.
- Notes**          Orbix specific.
- See Also**        `IT_daemon::addLaunchRights()`

### IT\_daemon::removeLaunchRightsDir()

- Synopsis**      `void IT_daemon::removeLaunchRightsDir(  
                  in string userGroup,  
                  in string dirName);`
- Description**    Removes the user or group in `userGroup` from the *launch* access control list for the directory `dirName`.
- Notes**          Orbix specific.
- See Also**        `IT_daemon::addLaunchRightsDir()`

### IT\_daemon::removeMethod()

- Synopsis**      `void IT_daemon::removeMethod(  
                  in string serverName,  
                  in string methodName);`
- Description**    Removes the activation order for the method (or method pattern) in `methodName` from the Implementation Repository entry for the per-method server, `serverName`.
- Notes**          Orbix specific.
- See Also**        `IT_daemon::addMethod()`

### IT\_daemon::removeSharedMarker()

- Synopsis**      `void IT_daemon::removeSharedMarker(  
                  in string serverName,  
                  in string markerName);`
- Description**    Removes the activation order for the marker (or marker pattern) in `markerName` from the Implementation Repository entry for the shared server, `serverName`.
- Notes**          Orbix specific.
- See Also**        `IT_daemon::addSharedMarker()`

### IT\_daemon::removeUnsharedMarker()

**Synopsis**

```
void IT_daemon::removeUnsharedMarker(
 in string serverName,
 in string markerName);
```

**Description**

Removes the activation order for the marker (or marker pattern) in `markerName` from the Implementation Repository entry for the unshared server, `serverName`.

**Notes**

Orbix specific.

**See Also**

`IT_daemon::addUnsharedMarker()`

### IT\_daemon::serverDetails

**Synopsis**

```
struct IT_daemon::serverDetails {
 string server;
 string marker;
 string principal;
 string code;
 string comms;
 string port;
 unsigned long OSspecific;
 LaunchStatus status;
};
```

**Description**

The members of the struct are as follows:

|            |                                                                                                                   |
|------------|-------------------------------------------------------------------------------------------------------------------|
| server     | The name of the server.                                                                                           |
| marker     | The marker (if any).                                                                                              |
| principal  | The name of the user for whom the server was launched. This is null if the server is not a per-client server.     |
| code       | The encoding protocol—for example, XDR.                                                                           |
| comms      | The transport protocol—for example, TCP/IP.                                                                       |
| port       | The port used by the communications system.                                                                       |
| OSspecific | On UNIX, OpenVMS and OS/390 this is an operating system process identifier of the server process.                 |
| status     | One of the enumerated values, <code>inactive</code> , <code>manualLaunch</code> or <code>automaticLaunch</code> . |

**Notes** Orbix specific.

**See Also** `IT_daemon::LaunchStatus`

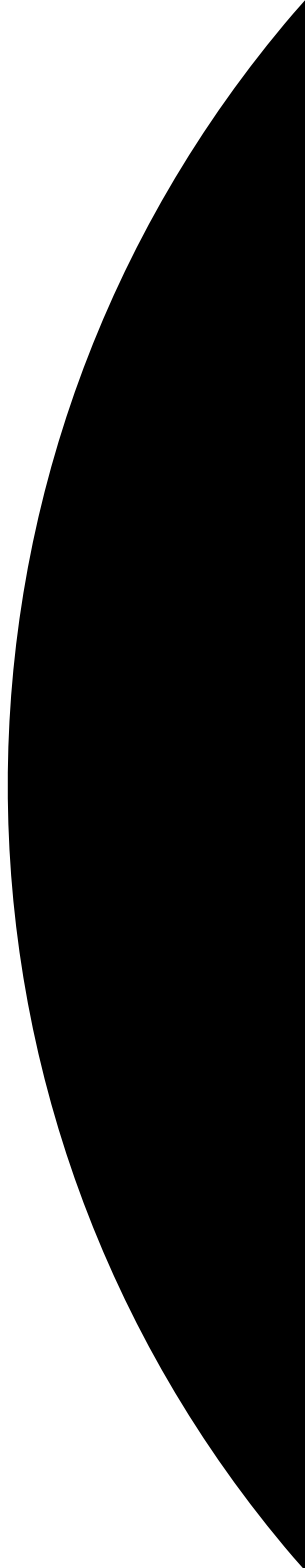
### **IT\_daemon::serverExists()**

**Synopsis** `CORBA::Boolean IT_daemon::serverExists(in string serverName);`

**Description** Determines whether there is an entry for the server `serverName` in the Implementation Repository.

**Notes** Orbix specific.

# Appendices







# Appendix A

## IDL Reference

This appendix presents reference material on the Interface Definition Language.

### IDL Grammar

This section presents the grammar of IDL. The notation is as follows:

| Symbol | Meaning                                                                                                                                       |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| ::=    | Is defined to be                                                                                                                              |
|        | Alternatively                                                                                                                                 |
| <text> | Non-terminal                                                                                                                                  |
| "text" | Literal                                                                                                                                       |
| *      | The preceding syntactic unit is to be repeated zero or more times.                                                                            |
| +      | The preceding syntactic unit is to be repeated one or more times.                                                                             |
| { }    | The enclosed syntactic units are to be grouped as a single syntactic unit (the following * is to be applied to the enclosed syntactic units). |
| [ ]    | The enclosed syntactic unit is optional (it may occur zero or one time).                                                                      |

Note that the two characters ">>" are always interpreted as a right shift operator. This means that a declaration of the form:

```
// IDL
typedef sequence<sequence<long> > sslong;
```

cannot be written without a white space between the two > characters:

```
// IDL
// Illegal
typedef sequence<sequence<long>> sslong;
```

The same restriction applies in C++ when declaring template classes.

### IDL Grammar: EBNF

- (1) <specification> ::= <definition>+
- (2) <definition> ::= <type\_dcl> ";"  
| <const\_dcl> ";"  
| <except\_dcl> ";"  
| <interface> ";"  
| <module> ";"
- (3) <module> ::= "module" <identifier>  
"{" <definition>+ "}"
- (4) <interface> ::= <interface\_dcl>  
| <forward\_dcl>
- (5) <interface\_dcl> ::= <interface\_header>  
"{" <interface-body> "}"
- (6) <forward\_dcl> ::= "interface" <identifier>
- (7) <interface\_header> ::= "interface" <identifier>  
[<inheritance\_spec>]
- (8) <interface\_body> ::= <export>\*
- (9) <export> ::= <type\_dcl> ";"  
| <const\_dcl> ";"  
| <except\_dcl> ";"  
| <attr\_dcl> ";"  
| <op\_dcl> ";"

|                         |                                                                                                                        |
|-------------------------|------------------------------------------------------------------------------------------------------------------------|
| (10) <inheritance_spec> | ::= ":" <scoped_name> {"," <scoped_name>}*                                                                             |
| (11) <scoped_name>      | ::= <identifier><br>  "::" <identifier><br>  <scoped_name> "::" <identifier>                                           |
| (12) <const_dcl>        | ::= "const" <const_type> <identifier><br>"="<const_exp>                                                                |
| (13) <const_type>       | ::= <integer_type><br>  <char_type><br>  <boolean_type><br>  <floating_pt_type><br>  <string_type><br>  <scoped_name>  |
| (14) <const_exp>        | ::= <or_expr>                                                                                                          |
| (15) <or_expr>          | ::= <xor_expr><br>  <or_expr> " " <xor_expr>                                                                           |
| (16) <xor_expr>         | ::= <and_expr><br>  <xor_expr> "^" <and_expr>                                                                          |
| (17) <and_expr>         | ::= <shift_expr><br>  <and_expr> "&" <shift_expr>                                                                      |
| (18) <shift_expr>       | ::= <add_expr><br>  <shift_expr> ">>"<add_expr><br>  <shift_expr> "<<"<add_expr>                                       |
| (19) <add_expr>         | ::= <mult_expr><br>  <add_expr> "+" <mult_expr><br>  <add_expr> "-" <mult_expr>                                        |
| (20) <mult_expr>        | ::= <unary_expr><br>  <mult_expr> "*" <unary_expr><br>  <mult_expr> "/" <unary_expr><br>  <mult_expr> "%" <unary_expr> |
| (21) <unary_expr>       | ::= <unary_operator> <primary_expr><br>  <primary_expr>                                                                |
| (22) <unary_operator>   | ::= "-"<br>  "+"<br>  "~"                                                                                              |

|                           |                                                                                                                        |
|---------------------------|------------------------------------------------------------------------------------------------------------------------|
| (23) <primary_expr>       | ::= <scoped_name><br>  <literal><br>  "(" <const_exp> ")"                                                              |
| (24) <literal>            | ::= <integer_literal><br>  <string_literal><br>  <character_literal><br>  <floating_pt_literal><br>  <boolean_literal> |
| (25) <boolean_literal>    | ::= "TRUE"<br>  "FALSE"                                                                                                |
| (26) <positive_int_const> | ::= <const_exp>                                                                                                        |
| (27) <type_dcl>           | ::= "typedef" <type_declarator><br>  <struct_type><br>  <union_type><br>  <enum_type>                                  |
| (28) <type_declarator>    | ::= <type_spec> <declarators>                                                                                          |
| (29) <type_spec>          | ::= <simple_type_spec><br>  <constr_type_spec>                                                                         |
| (30) <simple_type_spec>   | ::= <base_type_spec><br>  <template_type_spec><br>  <scoped_name>                                                      |
| (31) <base_type_spec>     | ::= <floating_pt_type><br>  <integer_type><br>  <char_type><br>  <boolean_type><br>  <octet_type><br>  <any_type>      |
| (32) <template_type_spec> | ::= <sequence_type><br>  <string_type>                                                                                 |
| (33) <constr_type_spec>   | ::= <struct_type><br>  <union_type><br>  <enum_type>                                                                   |
| (34) <declarators>        | ::= <declarator> { "," <declarator> }*                                                                                 |
| (35) <declarator>         | ::= <simple_declarator>                                                                                                |

|      |                                         |     |                                                           |
|------|-----------------------------------------|-----|-----------------------------------------------------------|
|      |                                         |     | <code>&lt;complex_declarator&gt;</code>                   |
| (36) | <code>&lt;simple_declarator&gt;</code>  | ::= | <code>&lt;identifier&gt;</code>                           |
| (37) | <code>&lt;complex_declarator&gt;</code> | ::= | <code>&lt;array_declarator&gt;</code>                     |
| (38) | <code>&lt;floating_pt_type&gt;</code>   | ::= | <code>"float"</code>                                      |
|      |                                         |     | <code>"double"</code>                                     |
| (39) | <code>&lt;integer_type&gt;</code>       | ::= | <code>&lt;signed_int&gt;</code>                           |
|      |                                         |     | <code>&lt;unsigned_int&gt;</code>                         |
| (40) | <code>&lt;signed_int&gt;</code>         | ::= | <code>&lt;signed_long_int&gt;</code>                      |
|      |                                         |     | <code>&lt;signed_short_int&gt;</code>                     |
| (41) | <code>&lt;signed_long_int&gt;</code>    | ::= | <code>"long"</code>                                       |
| (42) | <code>&lt;signed_short_int&gt;</code>   | ::= | <code>"short"</code>                                      |
| (43) | <code>&lt;unsigned_int&gt;</code>       | ::= | <code>&lt;unsigned_long_int&gt;</code>                    |
|      |                                         |     | <code>&lt;unsigned_short_int&gt;</code>                   |
| (44) | <code>&lt;unsigned_long_int&gt;</code>  | ::= | <code>"unsigned" "long"</code>                            |
| (45) | <code>&lt;unsigned_short_int&gt;</code> | ::= | <code>"unsigned" "short"</code>                           |
| (46) | <code>&lt;char_type&gt;</code>          | ::= | <code>"char"</code>                                       |
| (47) | <code>&lt;boolean_type&gt;</code>       | ::= | <code>"boolean"</code>                                    |
| (48) | <code>&lt;octet_type&gt;</code>         | ::= | <code>"octet"</code>                                      |
| (49) | <code>&lt;any_type&gt;</code>           | ::= | <code>"any"</code>                                        |
| (50) | <code>&lt;struct_type&gt;</code>        | ::= | <code>"struct" &lt;identifier&gt;</code>                  |
|      |                                         |     | <code>"{" &lt;member_list&gt; "}"</code>                  |
| (51) | <code>&lt;member_list&gt;</code>        | ::= | <code>&lt;member&gt;+</code>                              |
| (52) | <code>&lt;member&gt;</code>             | ::= | <code>&lt;type_spec&gt; &lt;declarators&gt; ";"</code>    |
| (53) | <code>&lt;union_type&gt;</code>         | ::= | <code>"union" &lt;identifier&gt; "switch"</code>          |
|      |                                         |     | <code>"(" &lt;switch_type_spec&gt; ")"</code>             |
|      |                                         |     | <code>"{" &lt;switch_body&gt; "}"</code>                  |
| (54) | <code>&lt;switch_type_spec&gt;</code>   | ::= | <code>&lt;integer_type&gt;</code>                         |
|      |                                         |     | <code>&lt;char_type&gt;</code>                            |
|      |                                         |     | <code>&lt;boolean_type&gt;</code>                         |
|      |                                         |     | <code>&lt;enum_type&gt;</code>                            |
|      |                                         |     | <code>&lt;scoped_name&gt;</code>                          |
| (55) | <code>&lt;switch_body&gt;</code>        | ::= | <code>&lt;case&gt;+</code>                                |
| (56) | <code>&lt;case&gt;</code>               | ::= | <code>&lt;case_label&gt;+ &lt;element_spec&gt; ";"</code> |
| (57) | <code>&lt;case_label&gt;</code>         | ::= | <code>"case" &lt;const_exp&gt; ":"</code>                 |

```
| "default" ":"
(58) <element_spec> ::= <type_spec> <declarator>
(59) <enum_type> ::= "enum" <identifier> "{" <enumerator>
 { "," <enumerator> }* "}"
(60) <enumerator> ::= <identifier>
(61) <sequence_type> ::= "sequence" "<" <simple_type_spec>
 "," <positive_int_const> ">"
| "sequence" "<" <simple_type_spec> ">"
(62) <string_type> ::= "string" "<" <positive_int_const> ">"
| "string"
(63) <array_declarator> ::= <identifier> <fixed_array_size>+
(64) <fixed_array_size> ::= "[" <positive_int_const> "]"
(65) <attr_dcl> ::= ["readonly"] "attribute"
 <param_type_spec>
 <simple_declarator>
 {","<simple_declarator>}*
(66) <except_dcl> ::= "exception" <identifier>
 "{" <member>* "}"
(67) <op_dcl> ::= [<op_attribute>] <op_type_spec>
 <identifier>
 <parameter_dcls>
 [<raises_expr>] [<context_expr>]
(68) <op_attribute> ::= "oneway"
(69) <op_type_spec> ::= <param_type_spec>
| "void"
(70) <parameter_dcls> ::= "(" <param_dcl> {"," <param_dcl>}* ")"
| "(" ")"
(71) <param_dcl> ::= <param_attribute> <param_type_spec>
 <simple_declarator>
(72) <param_attribute> ::= "in"
| "out"
| "inout"
(73) <raises_expr> ::= "raises" "(" <scoped_name>
 { "," <scoped_name> }* ")"
```

```
(74) <context_expr> ::= "context" "(" <string_literal>
 { "," <string_literal>* "}"
(75) <param_type_spec> ::= <base_type_spec> <string_type>
 <scoped_name>
```

## Keywords

The following are keywords in IDL.

|           |           |           |          |          |
|-----------|-----------|-----------|----------|----------|
| any       | default   | interface | readonly | unsigned |
| attribute | double    | long      | sequence | union    |
| boolean   | enum      | module    | short    | void     |
| case      | exception | octet     | string   | FALSE    |
| char      | float     | oneway    | struct   | Object   |
| const     | in        | out       | switch   | TRUE     |
| context   | inout     | raises    | typedef  |          |

Keywords must be written exactly as shown. For example, writing `Boolean` rather than `boolean` gives a compilation error.





# Appendix B

## System Exceptions

### System Exceptions Defined by CORBA

| Identifier | Exception     | Description                           |
|------------|---------------|---------------------------------------|
| 10000      | UNKNOWN       | Unknown exception                     |
| 10020      | BAD_PARAM     | An invalid parameter was passed       |
| 10040      | NO_MEMORY     | Dynamic memory allocation failure     |
| 10060      | IMP_LIMIT     | Violated implementation limit         |
| 10080      | COMM_FAILURE  | Communication failure                 |
| 10100      | INV_OBJREF    | Invalid object reference              |
| 10120      | NO_PERMISSION | No permission for attempted operation |
| 10140      | INTERNAL      | ORB internal error                    |
| 10160      | MARSHAL       | Error marshalling parameter/result    |
| 10180      | INITIALIZE    | ORB initialisation failure            |
| 10200      | NO_IMPLEMENT  | Operation implementation unavailable  |
| 10220      | BAD_TYPECODE  | Bad TypeCode                          |
| 10240      | BAD_OPERATION | Invalid operation                     |
| 10260      | NO_RESOURCES  | Insufficient resources for request    |
| 10280      | NO_RESPONSE   | Response to request not yet available |
| 10300      | PERSIST_STORE | Persistent storage failure            |

| Identifier | Exception       | Description                          |
|------------|-----------------|--------------------------------------|
| 10320      | BAD_INV_ORDER   | Routine invocations out of order     |
| 10340      | TRANSIENT       | Transient failure - reissue request  |
| 10360      | FREE_MEM        | Cannot free memory                   |
| 10380      | INV_IDENT       | Invalid identifier syntax            |
| 10400      | INV_FLAG        | Invalid flag was specified           |
| 10420      | INTF_REPOS      | Error accessing interface repository |
| 10440      | BAD_CONTEXT     | Error processing context object      |
| 10460      | OBJ_ADAPTOR     | Failure detected by object adaptor   |
| 10480      | DATA_CONVERSION | Data conversion error                |

### System Exceptions Specific to Orbix

| Identifier | Orbix Exception | Description                                                               |
|------------|-----------------|---------------------------------------------------------------------------|
| 10500      | FILTER_SUPPRESS | Suppress exception raised in per-object pre-filter                        |
| 10540      | ASCII_FILE      | ASCII file error                                                          |
| 10560      | LICENCING       | Licensing error                                                           |
| 10600      | IIOP            | IIOP error                                                                |
| 10620      | NO_CONFIG_VALUE | No configuration value set for one of the mandatory configuration entries |

---

# Index

## A

abortSlowConnects() 207  
absolute\_name() 352  
ActivateCVHandler() 208  
ActivateOutputHandler() 208  
activationMode 63  
active\_transactions() 380  
add() 163  
addDirRights() 410  
addForeignFD() 133, 208  
addForeignFDSet() 133, 209  
addInvokeRights() 410  
addInvokeRightsDir() 411  
add\_item() 163  
add\_item\_consume() 164  
addLaunchRights() 411  
addLaunchRightsDir() 411  
addMethod() 412  
addSharedMarker() 412  
addUnsharedMarker() 412  
add\_value() 164  
add\_value\_consume() 165  
~Any() 47  
Any() 46  
anyClientsConnected() 63  
arg() 35  
arguments() 277, 294  
assumeArgsOwnership() 278  
assumeResultOwnership() 278  
\_attachPost() 177  
\_attachPre() 178  
AuthenticationFilter() 55

## B

baseInterfacesOf() 209  
bindUsingIOP() 210  
BOA, initialisation 211  
BOA\_init() 211  
bound 393, 395

## C

change\_implementation() 63  
changeOwnerServer() 413

channels, closing 179  
clear() 101  
\_closeChannel() 179  
closeChannel() 212  
clrf() 128  
collocated() 212, 213  
commit() 380  
completed() 312  
CompletionStatus 313  
configuration 113  
    runtime 111  
connectionTimeout() 214  
containing\_repository() 352  
contents() 359  
    ~Context() 89  
Context() 88, 89  
    ~ContextIterator() 95  
ContextIterator() 95  
context\_name() 90  
contexts 384  
continueThreadDispatch() 64  
CORBA 222, 223, 224  
CORBA 35  
CORBA::ExceptionDef:  
    type 373  
CORBA:  
    arg() 35  
    CompletionStatus 313  
    default\_environment 36  
    extract() 36  
    insert() 37  
    is\_nil() 38  
    \_OBJECT\_TABLE\_SIZE\_DEFAULT 35  
    ORB\_init() 39  
    release() 40  
    string\_alloc() 40  
    string\_dup() 41  
    string\_free() 41  
CORBA::AliasDef 339  
CORBA::AliasDef:  
    describe() 339  
    original\_type\_def 340  
CORBA::Any 43  
CORBA::Any::

- ~Any() 47
- Any() 46
- operator<=<() 48
- operator=() 48
- operator>>=() 50
- replace() 51
- type() 52
- value() 53
- CORBA::ArrayDef 341
- CORBA::ArrayDef:
  - element\_type 341
  - element\_type\_def 342
  - length 342
- CORBA::AttributeDef 343
- CORBA::AttributeDef:
  - describe() 343
  - mode 344
  - type 344
  - type\_def 345
- CORBA::BOA:
  - activationMode 63
  - anyClientsConnected() 63
  - change\_implementation() 63
  - continueThreadDispatch() 64
  - create() 64
  - deactivate\_impl() 65
  - deactivate\_obj() 66
  - dispose() 66
  - enableLoaders() 67
  - filterBadConnectAttempts() 67, 68
  - getFileDescriptors() 71
  - getFilter() 71
  - get\_id() 68
  - get\_principal() 69, 74
  - impl\_is\_ready() 72
  - isEventPending() 75
  - myActivationMode() 75
  - myImplementationName() 75
  - myImpRepPath() 76
  - myIntRepPath() 76
  - myMarkerName() 76
  - myMarkerPattern() 77
  - myMethodName() 77
  - obj\_is\_ready() 77
  - processEvents() 78
  - processNextEvent() 80
  - propagateTIEdelete() 80
  - setImpl() 82
  - setNoHangup() 83
- CORBA::ConstantDef 347
- CORBA::ConstantDef:
  - describe() 347
  - type 348
  - type\_def 348
  - value 349
- CORBA::Contained 351
- CORBA::Contained:
  - absolute\_name() 352
  - containing\_repository() 352
  - defined\_in 352
  - describe() 353
  - id 354
  - move() 354
  - name() 355
  - version 355
- CORBA::Container 357
- CORBA::Container:
  - contents() 359
  - create\_alias() 360
  - create\_constant() 360
  - create\_enum() 361
  - create\_exception() 362
  - create\_interface() 363
  - create\_module() 363
  - create\_struct() 364
  - create\_union() 365
  - describe\_contents() 366
  - lookup() 366
  - lookup\_name() 367
- CORBA::Context 85
- CORBA::Context:
  - ~Context() 89
  - Context() 88, 89
  - context\_name() 90
  - create\_child() 90
  - delete\_values() 91
  - \_duplicate() 89
  - get\_count() 91
  - get\_count\_all() 91
  - get\_values() 92
  - IT\_create() 93
  - \_nil() 90
  - parent() 93
  - set\_one\_value() 94
  - set\_values() 94
- CORBA::ContextIterator 95
- CORBA::ContextIterator:
  - ~ContextIterator() 95
  - ContextIterator() 95
  - operator()() 96

- 
- CORBA::DefinitionKind 337
  - CORBA::DynamicImplementation 97
  - CORBA::DynamicImplementation::
    - ~DynamicImplementation() 98
    - DynamicImplementation() 97
    - invoke() 98
  - CORBA::EnumDef 369
  - CORBA::EnumDef::
    - describe() 369
    - members 370
  - CORBA::Environment 99
  - CORBA::Environment::
    - clear() 101
    - \_duplicate() 101
    - ~Environment() 103
    - Environment() 101, 102
    - exception() 103, 104
    - int() 104
    - IT\_create() 104
    - \_nil() 105
    - operator=() 105, 106
  - CORBA::Exception 109
  - CORBA::Exception::
    - ~Exception() 110
    - Exception() 109
    - operator=() 110
  - CORBA::ExceptionDef 371
  - CORBA::ExceptionDef::
    - describe() 371
    - members 372
  - CORBA::ExtraConfigFileCVHandler 111
  - CORBA::ExtraConfigFileCVHandler::
    - ~ExtraConfigFileCVHandler() 112
    - ExtraConfigFileCVHandler() 112
  - CORBA::ExtraRegistryCVHandler 113
  - CORBA::ExtraRegistryCVHandler::
    - ~ExtraRegistryCVHandler() 115
    - ExtraRegistryCVHandler() 114
    - GetRegKey() 115
  - CORBA::Filter 117
  - CORBA::Filter::
    - ~Filter() 118
    - Filter() 118
    - inReplyFailure() 118
    - inReplyPostMarshal() 119
    - inReplyPreMarshal() 120
    - inRequestPostMarshal() 120
    - inRequestPreMarshal() 121
    - outReplyFailure() 122
    - outReplyPostMarshal() 123
  - outReplyPreMarshal() 123
  - outRequestPostMarshal() 124
  - outRequestPreMarshal() 125
  - CORBA::Flags 127
  - CORBA::Flags::
    - clrfl() 128
    - Flags() 127, 128
    - isNil() 129
    - isSet() 129
    - isSetAll() 129
    - isSetAny() 129
    - operator=() 128
    - reset() 130
    - setArgDef() 130
    - setfl() 130
    - ULong() 130
  - CORBA::Identifier 337
  - CORBA::IDLType 375
  - CORBA::IDLType::
    - type 375
  - CORBA::ImplementationDef 131
  - CORBA::ImplementationDef::
    - \_duplicate() 131
    - IT\_create() 132
    - \_nil() 132
  - CORBA::IObject 377
  - CORBA::IObject::
    - def\_kind 377
    - destroy() 377
  - CORBA::IT\_IOCcallback 133
  - CORBA::IT\_IOCcallback::
    - ForeignFDExcept() 135
    - ForeignFDRead() 135
    - ForeignFDWrite() 135
    - OrbixFDclose() 136
    - OrbixFDOpen() 136
  - CORBA::IT\_Repository 379
  - CORBA::IT\_Repository::
    - active\_transactions() 380
    - commit() 380
    - start() 379
  - CORBA::IT\_Repository::rollBack() 380
  - CORBA::IT\_reqTransformer 133, 139
  - CORBA::IT\_reqTransformer::
    - free\_buf() 140
    - setRemoteHost() 143
    - transform() 141
    - transform\_error() 142
  - CORBA::LoaderClass 145
  - CORBA::LoaderClass::
    - outReplyPreMarshal() 123
    - outRequestPostMarshal() 124
    - outRequestPreMarshal() 125

- load() 147
- ~LoaderClass() 146
- LoaderClass() 146
- record() 148
- rename() 149
- save() 149
- CORBA::ModuleDef 381
- CORBA::ModuleDef::
  - describe() 381
- CORBA::NamedValue 151
- CORBA::NamedValue::
  - \_duplicate() 154
  - flags() 154
  - IT\_create() 153
  - name() 154
  - ~NamedValue() 153
  - NamedValue() 152
  - \_nil() 155
  - operator=() 155
  - value() 155
- CORBA::NullLoaderClass 157
- CORBA::NullLoaderClass::
  - NullLoaderClass() 157
  - record() 158
- CORBA::NVList 159
- CORBA::NVList::
  - add() 163
  - add\_item() 163
  - add\_item\_consume() 164
  - add\_value() 164
  - add\_value\_consume() 165
  - count() 165
  - \_duplicate() 162
  - IT\_create() 166
  - item() 166
  - \_nil() 163
  - ~NVList() 162
  - NVList() 161, 162
  - operator=() 162
  - remove() 167
- CORBA::NVListIterator 169
- CORBA::NVListIterator::
  - NVListIterator() 169, 170
  - operator()() 170
  - setList() 170
- CORBA::Object 171
- CORBA::Object::
  - \_attachPost() 177
  - \_attachPre() 178
  - \_closeChannel() 179
  - \_create\_request() 179
  - \_deref() 180
  - \_duplicate() 181
  - \_enableInternalLock() 182
  - \_fd() 183
  - \_get\_implementation() 183
  - \_get\_interface() 184
  - \_getPost() 184
  - \_getPre() 184
  - \_hash() 185
  - \_hasValidOpenChannel() 185
  - \_host() 186
  - \_implementation() 186
  - \_interfaceHost() 187
  - \_interfaceImplementation() 187
  - \_interfaceMarker() 187
  - \_is\_equivalent() 188
  - \_isNull() 189
  - \_isNullProxy() 189
  - \_isRemote() 190
  - \_loader() 190
  - \_marker() 191
  - \_nil() 192
  - \_non\_existent() 192
  - ~Object() 177
  - Object() 176
  - \_object\_to\_string() 192
  - \_refCount() 193
  - \_request() 193
  - \_save() 194
- CORBA::OperationDef 383
- CORBA::OperationDef::
  - contexts 384
  - describe() 384
  - exceptions 384
  - mode 385
  - params 385
  - result 386
  - result\_def 386
- CORBA::ORB::
  - abortSlowConnects() 207
  - ActivateCVHandler() 208
  - ActivateOutputHandler() 208
  - addForeignFD() 133, 208
  - addForeignFDSet() 133, 209
  - baseInterfacesOf() 209
  - bindUsingIIOP() 210
  - BOA\_init() 211
  - closeChannel() 212
  - collocated() 212, 213

- 
- connectionTimeout() 214
  - create\_environment() 215
  - create\_list() 215
  - create\_named\_value() 216
  - create\_operation\_list() 216
  - DeactivateCVHandler() 217
  - DeactivateOutputHandler() 217
  - DEFAULT\_TIMEOUT 218
  - defaultTxTimeout() 218, 219, 220, 227
  - eagerListeners() 220, 221
  - getAllOrbixFDs() 222
  - GetConfigValue() 222, 223, 224
  - get\_default\_context() 225
  - getForeignFDSet() 222
  - get\_next\_response() 225
  - getReqTransformer() 224
  - getSelectableFDSet() 224
  - INFINITE\_TIMEOUT 226
  - isBaseInterfaceOf() 226
  - isForeignFD() 227
  - isOrbixFD() 228
  - isOrbixSelectableFD() 228
  - list\_initial\_services() 229
  - makeIOR() 229
  - makeOrbixObjectKey() 230
  - maxConnectRetries() 230, 231
  - myHost() 232
  - myServer() 232
  - noReconnectOnFailure() 233, 234
  - object\_to\_string() 234
  - optimiseProtocolEncoding() 235, 236
  - Output() 236
  - pingDuringBind() 237
  - PlaceCVHandlerAfter() 238
  - PlaceCVHandlerBefore() 239
  - poll\_next\_response() 239
  - registerIOCallback() 240
  - registerIOCallbackObject() 241
  - ReinitialiseConfig() 243
  - removeForeignFD() 244
  - removeForeignFDSet() 244
  - reSizeObjectTable() 245
  - resolve\_initial\_references() 245
  - resortToStatic() 246
  - send\_multiple\_requests\_deferred() 247
  - send\_multiple\_requests\_oneway() 248
  - SetConfigValue() 249, 250
  - setDiagnostics() 251
  - setMyReqTransformer() 251
  - setReqTransformer() 252
  - setServerName() 252
  - set\_unsafeDelete() 253
  - string\_to\_object() 254, 255
  - unregisterIOCallbackObject() 257
  - useHostNameInIOR() 259
  - useTransientPort() 260, 261, 262, 263
  - CORBA::PrimitiveDef 387
  - CORBA::PrimitiveDef:
    - kind 387
  - CORBA::Principal 265
  - CORBA::Principal:
    - \_duplicate() 266
    - IT\_create() 266
    - \_nil() 266
    - Principal() 265
  - CORBA::Repository 389
  - CORBA::Repository:
    - create\_array() 390
    - create\_sequence() 390
    - create\_string() 391
    - describe\_contents() 391
    - get\_primitive() 391
    - lookup\_id() 392
  - CORBA::RepositoryId 338
  - CORBA::Request 267
  - CORBA::Request:
    - arguments() 277, 294
    - assumeOrigArgsOwnership() 278
    - assumeResultOwnership() 278
    - ctx() 279
    - decodeArray() 279
    - descriptor() 282
    - \_duplicate() 277
    - encodeArray() 282
    - env() 283
    - extractOctet() 283
    - get\_response() 284
    - insertOctet() 284
    - invoke() 285
    - IT\_create() 285
    - \_nil() 277
    - operation() 286
    - operator<<() 275
    - operator>>() 274
    - poll\_response() 286
    - ~Request() 273
    - Request() 272, 273, 274
    - reset() 287
    - result() 287
    - send\_deferred() 287

- send\_oneway() 288
- setOperation() 289
- set\_return\_type() 290
- setTarget() 290
- target() 290
- CORBA::ScopedName 338
- CORBA::SequenceDef 393
- CORBA::SequenceDef::
  - bound 393
  - element\_type 394
  - element\_type\_def 394
  - type 394
- CORBA::ServerRequest 291
- CORBA::ServerRequest::
  - ctx() 294
  - env() 294, 295
  - op\_def() 295
  - operation() 296
  - op\_name() 296
  - params() 297
  - ~Request() 293
  - result() 297
  - target() 298
- CORBA::StringDef 395
- CORBA::StringDef::
  - bound 395
- CORBA::String\_var 305
- CORBA::String\_var::
  - char\*() 307
  - operator=() 307
  - ~String\_var() 306
  - String\_var() 306
- CORBA::StructDef 397
- CORBA::StructDef::
  - members 398
- CORBA::StructDef::describe() 397
- CORBA::SystemException 309
- CORBA::SystemException::
  - completed() 312
  - minor() 313
  - \_narrow() 312
  - operator-() 311
  - operator<<() 311
  - ~SystemException() 311
  - SystemException() 310, 311
- CORBA::ThreadFilter 315
- CORBA::ThreadFilter::
  - ThreadFilter() 316
- CORBA::TypeCode 317
- CORBA::TypeCode::
  - \_duplicate() 323
  - equal() 324
  - IT\_create() 324
  - kind() 324
  - \_nil() 323
  - operator!=() 323
  - operator=() 322
  - operator==(()) 322
  - param\_count() 325
  - parameter() 325
  - ~TypeCode() 322
  - TypeCode() 321, 322
- CORBA::TypedefDef 399
- CORBA::TypedefDef::
  - describe() 399
- CORBA::UnionDef 401
- CORBA::UnionDef::
  - describe() 401
  - discriminator\_type() 402
  - discriminator\_type\_def() 402
  - members 403
- CORBA::UserCVHandler 299, 327
- CORBA::UserCVHandler::
  - GetValue() 300, 301, 302, 303, 328
  - ~UserCVHandler() 328
  - UserCVHandler() 299, 300, 301, 302, 303, 328
- CORBA::UserException 331
- CORBA::UserException::
  - \_narrow() 332
  - operator =() 332
  - UserException() 331, 332
- CORBA::UserOutput 333
- CORBA::UserOutput::
  - Output() 334
  - ~UserOutput() 334
  - UserOutput() 334
- count() 165
- create() 64
- create\_alias() 360
- create\_array() 390
- create\_child() 90
- create\_constant() 360
- create\_enum() 361
- create\_environment() 215
- create\_exception() 362
- create\_interface() 363
- create\_list() 215
- create\_module() 363
- create\_named\_value() 216
- create\_operation\_list() 216



---

\_create\_request() 179  
create\_sequence() 390  
create\_string() 391  
create\_struct() 364  
create\_union() 365  
ctx() 279, 294

**D**

Daemon  
  IDL definition 407  
DeactivateCVHandler() 217  
deactivate\_impl() 65  
deactivate\_obj() 66  
DeactivateOutputHandler() 217  
decodeArray() 279  
default\_environment 36  
DEFAULT\_TIMEOUT 218  
defaultTxTimeout() 218, 219, 220, 227  
defined\_in 352  
def\_kind 377  
deleteDirectory() 413  
deleteServer() 413  
delete\_values() 91  
\_deref() 180  
describe() 339, 343, 347, 353, 369, 371, 381, 384,  
  397, 399, 401  
describe\_contents() 366, 391  
descriptor() 282  
destroy() 377  
discriminator\_type() 402  
discriminator\_type\_def() 402  
dispose() 66  
\_duplicate() 89, 101, 131, 154, 162, 181, 266, 277,  
  323  
~DynamicImplementation() 98  
DynamicImplementation() 97

**E**

eagerListeners() 220, 221  
element\_type 341, 394  
element\_type\_def 342, 394  
\_enableInternalLock() 182  
enableLoaders() 67  
encodeArray() 282  
env() 283, 294, 295  
~Environment() 103  
Environment() 101, 102  
equal() 324  
~Exception() 110

Exception() 109  
exception() 103, 104  
exceptions 384  
~ExtraConfigFileCVHandler() 112  
ExtraConfigFileCVHandler() 112  
extract() 36  
extractOctet() 283  
~ExtraRegistryCVHandler() 115  
ExtraRegistryCVHandler() 114

**F**

\_fd() 183  
~Filter() 118  
Filter() 118  
filterBadConnectAttempts() 67, 68  
Flags() 127, 128  
flags() 154  
free\_buf() 140

**G**

getAllOrbixFDs() 222  
GetConfigValue() 222, 223, 224  
get\_count() 91  
get\_count\_all() 91  
get\_default\_context() 225  
getFileDescriptors() 71  
getFilter() 71  
getForeignFDSet() 222  
get\_id() 68  
\_get\_implementation() 183  
\_get\_interface() 184  
get\_next\_response() 225  
\_getPost() 184  
\_getPre() 184  
get\_primitive() 391  
get\_principal() 69, 74  
GetRegKey() 115  
getReqTransformer() 224  
get\_response() 284  
getSelectableFDSet() 224  
getServer() 414  
GetValue() 300, 301, 302, 303, 328  
get\_values() 92

**H**

Handlers  
  output 333  
\_hash() 185  
\_hasValidOpenChannel() 185

\_host() 186

## I

id 354

IDL definition

Implementation Repository 407

Orbix daemon 407

\_implementation() 186

impl\_is\_ready() 72

include files

daemon.hh 407

INFINITE\_TIMEOUT 226

initial services 229

initialisation

BOA 211

inReplyFailure() 118

inReplyPostMarshal() 119

inReplyPreMarshal() 120

inRequestPostMarshal() 120

inRequestPreMarshal() 121

insert() 37

insertOctet() 284

int() 104

\_interfaceHost() 187

\_interfaceImplementation() 187

\_interfaceMarker() 187

InterfaceName 391

invoke() 98, 285

isBaseInterfaceOf() 226

\_is\_equivalent() 188

isEventPending() 75

isForeignFD() 227

isNil() 129

is\_nil() 38

\_isNull() 189

\_isNullProxy() 189

isOrbixFD() 228

\_isRemote() 190

isSet() 129

isSetAll() 129

isSetAny() 129

IT\_create() 93, 104, 132, 153, 166, 266, 285, 324

IT\_daemon 407

IT\_daemon::

addDirRights() 410

addInvokeRights() 410

addInvokeRightsDir() 411

addLaunchRights() 411

addLaunchRightsDir() 411

addMethod() 412

addSharedMarker() 412

addUnsharedMarker() 412

changeOwnerServer() 413

deleteDirectory() 413

deleteServer() 413

getServer() 414

killServer() 414

LaunchStatus 415

listActiveServers() 415

listServers() 415

newDirectory() 416

newPerMethodServer() 416

newSharedServer() 417

newUnSharedServer() 418

removeDirRights() 418

removeInvokeRights() 419

removeInvokeRightsDir() 419

removeLaunchRights() 419

removeLaunchRightsDir() 420

removeMethod() 420

removeSharedMarker() 420

removeUnsharedMarker() 421

serverDetails 421

serverExists() 422

item() 166

## K

killServer() 414

kind 387

kind() 324

## L

LaunchStatus 415

length 342

listActiveServers() 415

list\_initial\_services() 229

listServers() 415

load() 147

\_loader() 190

~LoaderClass() 146

LoaderClass() 146

lookup() 366

lookup\_id() 392

lookup\_name() 367

## M

makeIOR() 229

makeOrbixObjectKey() 230

\_marker() 191

---

maxConnectRetries() 230, 231  
max\_returned\_objs 391  
members 370, 372, 398, 403  
minor() 313  
mode 344, 385  
move() 354  
myActivationMode() 75  
myHost() 232  
myImplementationName() 75  
myImpRepPath() 76  
myIntRepPath() 76  
myMarkerName() 76  
myMarkerPattern() 77  
myMethodName() 77  
myServer() 232

## N

name() 154, 355  
~NamedValue() 153  
NamedValue() 152  
\_narrow() 332  
newDirectory() 416  
newPerMethodServer() 416  
newSharedServer() 417  
newUnSharedServer() 418  
\_nil() 90, 105, 132, 155, 163, 192, 266, 277, 323  
\_non\_existent() 192  
noReconnectOnFailure() 233, 234  
NullLoaderClass() 157  
~NVList() 162  
NVList() 161, 162  
NVListIterator() 169, 170

## O

~Object() 177  
Object() 176  
\_OBJECT\_TABLE\_SIZE\_DEFAULT 35  
\_object\_to\_string() 192  
object\_to\_string() 234  
obj\_is\_ready() 77  
op\_def() 295  
operation() 286, 296  
operator!=() 323  
operator() 96, 170  
operator<<() 275  
operator<<=() 48  
operator=() 48, 105, 106, 110, 128, 155, 162, 307, 322  
operator==( ) 322

operator>>() 274  
operator>>=() 50  
op\_name() 296  
optimiseProtocolEncoding() 235, 236  
ORB\_init() 39  
orbixd, IDL definition for 407  
original\_type\_def 340  
Output Handlers 333  
Output() 236, 334  
outReplyFailure() 122  
outReplyPostMarshal() 123  
outReplyPreMarshal() 123  
outRequestPostMarshal() 124  
outRequestPreMarshal() 125

## P

param\_count() 325  
parameter() 325  
params 385  
params() 297  
parent() 93  
pingDuringBind() 237  
PlaceCVHandlerAfter() 238  
PlaceCVHandlerBefore() 239  
poll\_next\_response() 239  
poll\_response() 286  
Principal() 265  
processEvents() 78  
processNextEvent() 80  
propagateTIEdelete() 80

## R

record() 148, 158  
\_refCount() 193  
registerIOCallback() 240  
registerIOCallbackObject() 241  
registry 113  
ReinitialiseConfig() 243  
release() 40  
remove() 167  
removeDirRights() 418  
removeForeignFD() 242, 243, 244, 258  
removeForeignFDSet() 244  
removeInvokeRights() 419  
removeInvokeRightsDir() 419  
removeLaunchRights() 419  
removeLaunchRightsDir() 420  
removeMethod() 420  
removeSharedMarker() 420

removeUnsharedMarker() 421  
rename() 149  
replace() 51  
~Request() 273, 293  
Request() 272, 273, 274, 293  
\_request() 193  
reset() 130, 287  
reSizeObjectTable() 245  
resolve\_initial\_references() 245  
resortToStatic() 246  
result 386  
result() 287, 297  
rollBack() 380

### S

\_save() 194  
save() 149  
send\_deferred() 287  
send\_multiple\_requests\_deferred() 247  
send\_multiple\_requests\_oneway() 248  
send\_oneway() 288  
serverDetails 421  
serverExists() 422  
setArgDef() 130  
SetConfigValue() 249, 250  
setDiagnostics() 251  
setf() 130  
setImpl() 82  
setList() 170  
setMyReqTransformer() 251  
setNoHangup() 83  
set\_one\_value() 94  
setOperation() 289  
setRemoteHost() 143  
setReqTransformer() 252  
set\_return\_type() 290  
setServerName() 252  
setTarget() 290  
set\_unsafeDelete() 253  
set\_values() 94  
start() 379  
string\_alloc() 40  
string\_dup() 41  
string\_free() 41  
string\_to\_object() 254, 255  
~String\_var() 306  
String\_var() 306  
system registry 113  
SystemException() 310, 311, 312

### T

target() 290, 298  
ThreadFilter() 316  
transform() 141  
transform\_error() 142  
type 344, 348, 373, 375, 394  
type() 52  
~TypeCode() 322  
TypeCode() 321, 322  
type\_def 345, 348

### U

ULong() 130  
unregisterIOCallbackObject() 257  
useHostNameInIOR() 259, 260, 261, 262, 263  
~UserCVHandler() 328  
UserCVHandler() 299, 300, 301, 302, 303, 328  
UserException() 331, 332  
~UserOutput() 334  
UserOutput() 334  
useTransientPort() 261, 262

### V

value 349  
value() 53, 155  
version 355