

Orbix C++ Programmer's Reference

Orbix is a Registered Trademark of IONA Technologies PLC.

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

Java is a trademark of Sun Microsystems, Inc.

COPYRIGHT NOTICE

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

Copyright © 1991-1999 by IONA Technologies PLC. All rights reserved.

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

M 2 2 3 I

Contents

Preface	xxi
Audience	xxi
Organization of this Guide	xxi
Document Conventions	xxii

Part I Orbix Class Reference

CORBA	7
CORBA::_LOCATOR_HOPS	7
CORBA::_MAX_LOCATOR_HOPS	7
CORBA::_OBJECT_TABLE_SIZE_DEFAULT	8
CORBA::arg()	8
CORBA::default_environment	9
CORBA::extract()	9
CORBA::insert()	10
CORBA::is_nil()	11
CORBA::ORB_init()	12
CORBA::release()	13
CORBA::string_alloc()	13
CORBA::string_dup()	14
CORBA::string_free()	14
CORBA::Any	15
CORBA::Any::Any()	18
CORBA::Any::Any()	18
CORBA::Any::Any()	18
CORBA::Any::~~Any()	19
CORBA::Any::operator=()	20
CORBA::Any::operator<<=()	20
CORBA::Any::operator>>=()	22
CORBA::Any::replace()	23
CORBA::Any::type()	24
CORBA::Any::value()	25

CORBA::AuthenticationFilter	27
CORBA::AuthenticationFilter::AuthenticationFilter()	27
CORBA::BOA	29
CORBA::BOA::activationMode	33
CORBA::BOA::anyClientsConnected()	33
CORBA::BOA::change_implementation()	33
CORBA::BOA::continueThreadDispatch()	34
CORBA::BOA::create()	34
CORBA::BOA::deactivate_impl()	35
CORBA::BOA::deactivate_obj()	36
CORBA::BOA::dispose()	36
CORBA::BOA::enableLoaders()	37
CORBA::BOA::filterBadConnectAttempts()	37
CORBA::BOA::get_id()	38
CORBA::BOA::get_principal()	38
CORBA::BOA::getFileDescriptors()	40
CORBA::BOA::getFilter()	40
CORBA::BOA::impl_is_ready()	41
CORBA::BOA::isEventPending()	43
CORBA::BOA::myActivationMode()	44
CORBA::BOA::myImplementationName()	44
CORBA::BOA::myImpRepPath()	44
CORBA::BOA::myIntRepPath()	45
CORBA::BOA::myMarkerName()	45
CORBA::BOA::myMarkerPattern()	45
CORBA::BOA::myMethodName()	46
CORBA::BOA::obj_is_ready()	46
CORBA::BOA::processEvents()	47
CORBA::BOA::processNextEvent()	48
CORBA::BOA::propagateTIEdelete()	49
CORBA::BOA::setImpl()	50
CORBA::BOA::setNoHangup()	51
CORBA::Context	53
CORBA::Context::Context()	56
CORBA::Context::Context()	56
CORBA::Context::Context()	57
CORBA::Context::~Context()	57
CORBA::Context::_duplicate()	57
CORBA::Context::_nil()	58

CORBA::Context::context_name()	58
CORBA::Context::create_child()	58
CORBA::Context::delete_values()	59
CORBA::Context::get_count()	59
CORBA::Context::get_count_all()	59
CORBA::Context::get_values()	60
CORBA::Context::IT_create()	61
CORBA::Context::parent()	61
CORBA::Context::set_one_value()	62
CORBA::Context::set_values()	62
CORBA::ContextIterator	63
CORBA::ContextIterator::ContextIterator()	63
CORBA::ContextIterator::~~ContextIterator()	63
CORBA::ContextIterator::operator>()	64
CORBA::DynamicImplementation	65
CORBA::DynamicImplementation::DynamicImplementation()	65
CORBA::DynamicImplementation::~~DynamicImplementation()	66
CORBA::DynamicImplementation::invoke()	66
CORBA::Environment	67
CORBA::Environment::Environment()	69
CORBA::Environment::Environment()	69
CORBA::Environment::Environment()	69
CORBA::Environment::Environment()	70
CORBA::Environment::~~Environment()	70
CORBA::Environment::operator=()	70
CORBA::Environment::operator=()	71
CORBA::Environment::operator=()	71
operator<<()	71
CORBA::Environment::_duplicate()	71
CORBA::Environment::_nil()	72
CORBA::Environment::clear()	72
CORBA::Environment::exception()	73
CORBA::Environment::exception()	73
CORBA::Environment::int()	74
CORBA::Environment::IT_create()	74
CORBA::Environment::m_request	75
CORBA::Environment::propagate()	75
CORBA::Environment::timeout()	76

CORBA::Environment::timeout()	76
CORBA::Exception	77
CORBA::Exception::Exception()	77
CORBA::Exception::Exception()	77
CORBA::Exception::~~Exception()	78
CORBA::Exception::operator=()	78
CORBA::	
ExtraConfigFileCVHandler	79
CORBA::ExtraConfigFileCVHandler::ExtraConfigFileCVHandler()	80
CORBA::ExtraConfigFileCVHandler::~~ExtraConfigFileCVHandler()	80
CORBA::ExtraRegistryCVHandler	81
CORBA::ExtraRegistryCVHandler::ExtraRegistryCVHandler()	82
CORBA::ExtraRegistryCVHandler::ExtraRegistryCVHandler()	82
CORBA::ExtraRegistryCVHandler::~~ExtraRegistryCVHandler()	83
CORBA::ExtraRegistryCVHandler::GetRegKey()	83
CORBA::Filter	85
CORBA::Filter::Filter()	86
CORBA::Filter::~~Filter()	86
CORBA::Filter::inReplyFailure()	86
CORBA::Filter::inReplyPostMarshal()	87
CORBA::Filter::inReplyPreMarshal()	88
CORBA::Filter::inRequestPostMarshal()	88
CORBA::Filter::inRequestPreMarshal()	89
CORBA::Filter::outReplyFailure()	90
CORBA::Filter::outReplyPostMarshal()	91
CORBA::Filter::outReplyPreMarshal()	91
CORBA::Filter::outRequestPostMarshal()	92
CORBA::Filter::outRequestPreMarshal()	93
CORBA::Flags	95
CORBA::Flags::Flags()	95
CORBA::Flags::Flags()	96
CORBA::Flags::Flags()	96
CORBA::Flags::operator=()	96
CORBA::Flags::clrf()	96
CORBA::Flags::isNil()	97

CORBA::Flags::isSet()	97
CORBA::Flags::isSetAll()	97
CORBA::Flags::isSetAny()	97
CORBA::Flags::ULong()	98
CORBA::Flags::reset()	98
CORBA::Flags::setArgDef()	98
CORBA::Flags::setf()	98
CORBA::ImplementationDef	99
CORBA::ImplementationDef::_duplicate()	99
CORBA::ImplementationDef::_nil()	100
CORBA::ImplementationDef::IT_create()	100
CORBA::IT_IOCallback	101
CORBA::IT_IOCallback::ForeignFDExcept()	102
CORBA::IT_IOCallback::ForeignFDRead()	102
CORBA::IT_IOCallback::ForeignFDWrite()	102
CORBA::IT_IOCallback::OrbixFDClose()	103
CORBA::IT_IOCallback::OrbixFDOpen()	103
CORBA::IT_reqTransformer	105
CORBA::IT_reqTransformer::free_buf()	106
CORBA::IT_reqTransformer::transform()	107
CORBA::IT_reqTransformer::transform_error()	108
CORBA::IT_reqTransformer::setRemoteHost()	109
CORBA::ORB::getMyReqTransformer()	109
CORBA::ORB::setMyReqTransformer()	109
CORBA::ORB::setReqTransformer()	110
CORBA::LoaderClass	111
CORBA::LoaderClass::LoaderClass()	112
CORBA::LoaderClass::~LoaderClass()	112
CORBA::LoaderClass::load()	113
CORBA::LoaderClass::record()	114
CORBA::LoaderClass::rename()	115
CORBA::LoaderClass::save()	115
CORBA::locatorClass	117
CORBA::locatorClass::locatorClass()	117
CORBA::locatorClass::lookUp()	118

CORBA::NamedValue	119
CORBA::NamedValue::NamedValue()	120
CORBA::NamedValue::NamedValue()	120
CORBA::NamedValue::~~NamedValue()	121
CORBA::NamedValue::operator=()	121
CORBA::NamedValue::_duplicate()	121
CORBA::NamedValue::_nil()	121
CORBA::NamedValue::flags()	122
CORBA::NamedValue::IT_create()	122
CORBA::NamedValue::name()	123
CORBA::NamedValue::value()	123
CORBA::NullLoaderClass	125
CORBA::NullLoaderClass::NullLoaderClass()	125
CORBA::NullLoaderClass::record()	126
CORBA::NVList	127
CORBA::NVList::NVList()	129
CORBA::NVList::NVList()	129
CORBA::NVList::NVList()	129
CORBA::NVList::~~NVList()	130
CORBA::NVList::operator=()	130
CORBA::NVList::_duplicate()	130
CORBA::NVList::_nil()	130
CORBA::NVList::add()	131
CORBA::NVList::add_item()	131
CORBA::NVList::add_value()	132
CORBA::NVList::add_item_consume()	132
CORBA::NVList::add_value_consume()	133
CORBA::NVList::count()	133
CORBA::NVList::IT_create()	133
CORBA::NVList::item()	134
CORBA::NVList::remove()	134
CORBA::NVListIterator	135
CORBA::NVListIterator::NVListIterator()	135
CORBA::NVListIterator::NVListIterator()	136
CORBA::NVListIterator::operator>()()	136
CORBA::NVListIterator::setList()	136

CORBA::Object	137
CORBA::Object::Object()	140
CORBA::Object::Object()	141
CORBA::Object::Object()	141
CORBA::Object::Object()	141
CORBA::Object::Object()	142
CORBA::Object::~~Object()	142
CORBA::Object::operator=()	143
CORBA::Object::_attachPost()	143
CORBA::Object::_attachPre()	144
CORBA::Object::_closeChannel()	145
CORBA::Object::_create_request()	145
CORBA::Object::_deref()	146
CORBA::Object::_duplicate()	148
CORBA::Object::_enableInternalLock()	148
CORBA::Object::_fd()	149
CORBA::Object::_get_implementation()	150
CORBA::Object::_get_interface()	150
CORBA::Object::_getPost()	151
CORBA::Object::_getPre()	151
CORBA::Object::_hash()	151
CORBA::Object::_hasValidOpenChannel()	152
CORBA::Object::_host()	152
CORBA::Object::_implementation()	153
CORBA::Object::_interfaceHost()	153
CORBA::Object::_interfaceImplementation()	153
CORBA::Object::_interfaceMarker()	154
CORBA::Object::_is_a()	154
CORBA::Object::_is_equivalent()	155
CORBA::Object::_isNull()	155
CORBA::Object::_isNullProxy()	156
CORBA::Object::_isRemote()	156
CORBA::Object::_loader()	157
CORBA::Object::_marker()	157
CORBA::Object::_marker()	157
CORBA::Object::_nil()	158
CORBA::Object::_non_existent()	158
CORBA::Object::_object_to_string()	159
CORBA::Object::_refCount()	159
CORBA::Object::_request()	160
CORBA::Object::_save()	160

CORBA::ORB	161
CORBA::ORB::abortSlowConnects()	167
CORBA::ORB::addForeignFD()	168
CORBA::ORB::addForeignFDSet()	168
CORBA::ORB::ActivateCVHandler()	169
CORBA::ORB::ActivateOutputHandler()	169
CORBA::ORB::baseInterfacesOf()	170
CORBA::ORB::bindUsingIOP()	170
CORBA::ORB::bindUsingIOP()	170
CORBA::ORB::BOA_init()	171
CORBA::ORB::closeChannel()	172
CORBA::ORB::collocated()	173
CORBA::ORB::collocated()	173
CORBA::ORB::connectionTimeout()	174
CORBA::ORB::create_environment()	174
CORBA::ORB::create_list()	175
CORBA::ORB::create_named_value()	175
CORBA::ORB::create_operation_list()	176
CORBA::ORB::DeactivateCVHandler()	177
CORBA::ORB::DeactivateOutputHandler()	177
CORBA::ORB::DEFAULT_TIMEOUT	177
CORBA::ORB::defaultTxTimeout()	178
CORBA::ORB::eagerListeners()	178
CORBA::ORB::eagerListeners()	179
CORBA::ORB::getAllOrbixFDs()	179
CORBA::ORB::get_default_context()	180
CORBA::ORB::getForeignFDSet()	180
CORBA::ORB::get_next_response()	181
CORBA::ORB::getSelectableFDSet()	181
CORBA::ORB::GetConfigValue()	182
CORBA::ORB::INFINITE_TIMEOUT	182
CORBA::ORB::isBaseInterfaceOf()	183
CORBA::ORB::isForeignFD()	183
CORBA::ORB::isOrbixFD()	184
CORBA::ORB::isOrbixSelectableFD()	184
CORBA::ORB::list_initial_services()	185
CORBA::ORB::makeIOR()	185
CORBA::ORB::makeOrbixObjectKey()	186
CORBA::ORB::maxConnectionThreads()	186
CORBA::ORB::maxConnectionThreads()	187
CORBA::ORB::maxConnectRetries()	187
CORBA::ORB::maxFDsPerConnectionThread()	188

CORBA::ORB::maxFDsPerConnectionThread()	188
CORBA::ORB::myHost()	188
CORBA::ORB::myServer()	189
CORBA::ORB::nativeExceptions()	189
CORBA::ORB::nativeExceptions()	189
CORBA::ORB::noReconnectOnFailure()	190
CORBA::ORB::object_to_string()	191
CORBA::ORB::optimiseProtocolEncoding()	192
CORBA::ORB::Output()	192
CORBA::ORB::pingDuringBind()	193
CORBA::ORB::PlaceCVHandlerAfter()	193
CORBA::ORB::PlaceCVHandlerBefore()	194
CORBA::ORB::poll_next_response()	195
CORBA::ORB::registerIOCallback()	195
CORBA::ORB::registerIOCallbackObject()	197
CORBA::ORB::removeForeignFD()	198
CORBA::ORB::removeForeignFDSet()	198
CORBA::ORB::ReinitialiseConfig()	199
CORBA::ORB::reSizeObjectTable()	199
CORBA::ORB::resolve_initial_references()	200
CORBA::ORB::resortToStatic()	201
CORBA::ORB::send_multiple_requests_deferred()	202
CORBA::ORB::send_multiple_requests_oneway()	202
CORBA::ORB::set_unsafeDelete()	203
CORBA::ORB::SetConfigValue()	204
CORBA::ORB::setDiagnostics()	205
CORBA::ORB::setServerName()	205
CORBA::ORB::string_to_object()	206
CORBA::ORB::supportBidirectionalIOP()	206
CORBA::ORB::string_to_object()	207
CORBA::ORB::unregisterIOCallbackObject()	208
CORBA::ORB::useTransientPort()	209
CORBA::ORB::useHostNameInIOR()	209
CORBA::Principal	211
CORBA::Principal::Principal()	211
CORBA::Principal::_duplicate()	212
CORBA::Principal::_nil()	212
CORBA::Principal::IT_create()	212

CORBA::Request	213
CORBA::Request::Request()	217
CORBA::Request::Request()	217
CORBA::Request::~~Request()	218
CORBA::Request::operator>>()	218
CORBA::Request::operator<<()	219
CORBA::Request::_duplicate()	220
CORBA::Request::_nil()	221
CORBA::Request::arguments()	221
CORBA::Request::assumeArgsOwnership()	221
CORBA::Request::assumeResultOwnership()	222
CORBA::Request::ctx()	222
CORBA::Request::ctx()	222
CORBA::Request::decodeArray()	223
CORBA::Request::descriptor()	224
CORBA::Request::encodeArray()	224
CORBA::Request::env()	225
CORBA::Request::extractOctet()	225
CORBA::Request::get_response()	225
CORBA::Request::insertOctet()	226
CORBA::Request::invoke()	226
CORBA::Request::IT_create()	227
CORBA::Request::operation()	227
CORBA::Request::poll_response()	228
CORBA::Request::reset()	228
CORBA::Request::result()	229
CORBA::Request::send_deferred()	229
CORBA::Request::send_oneway()	230
CORBA::Request::setOperation()	230
CORBA::Request::set_return_type()	231
CORBA::Request::setTarget()	231
CORBA::Request::target()	231
CORBA::ServerRequest	233
CORBA::ServerRequest::ServerRequest()	235
CORBA::ServerRequest::~~ServerRequest()	235
CORBA::ServerRequest::arguments()	235
CORBA::ServerRequest::ctx()	236
CORBA::ServerRequest::exception()	236
CORBA::ServerRequest::env()	237
CORBA::ServerRequest::env()	237
CORBA::ServerRequest::op_def()	237

CORBA::ServerRequest::op_name()	238
CORBA::ServerRequest::operation()	238
CORBA::ServerRequest::params()	239
CORBA::ServerRequest::result()	239
CORBA::ServerRequest::target()	240
CORBA::String_var	241
CORBA::String_var::String_var()	242
CORBA::String_var::String_var()	242
CORBA::String_var::String_var()	242
CORBA::String_var::~~String_var()	242
CORBA::String_var::operator=()	243
CORBA::String_var::operator[]()	243
CORBA::String_var::char*()	243
CORBA::SystemException	245
CORBA::SystemException::SystemException()	246
CORBA::SystemException::SystemException()	247
CORBA::SystemException::SystemException()	247
CORBA::SystemException::~~SystemException()	247
CORBA::SystemException::operator=()	247
operator<<()	248
CORBA::SystemException::_narrow()	248
CORBA::SystemException::completed()	248
CORBA::SystemException::completed()	249
CORBA::CompletionStatus	249
CORBA::SystemException::minor()	249
CORBA::SystemException::minor()	250
CORBA::ThreadFilter	251
CORBA::ThreadFilter::ThreadFilter()	252
CORBA::TypeCode	253
CORBA::TypeCode::TypeCode()	257
CORBA::TypeCode::TypeCode()	258
CORBA::TypeCode::~~TypeCode()	258
CORBA::TypeCode::operator=()	258
CORBA::TypeCode::operator==(())	258
CORBA::TypeCode::operator!=(())	259
CORBA::TypeCode::_duplicate()	259

<code>CORBA::TypeCode::_nil()</code>	259
<code>CORBA::TypeCode::equal()</code>	260
<code>CORBA::TypeCode::IT_create()</code>	260
<code>CORBA::TypeCode::kind()</code>	260
<code>CORBA::TypeCode::param_count()</code>	261
<code>CORBA::TypeCode::parameter()</code>	261
CORBA::UserCVHandler	263
<code>CORBA::UserCVHandler::UserCVHandler()</code>	264
<code>CORBA::UserCVHandler::~~UserCVHandler()</code>	264
<code>CORBA::UserCVHandler::GetValue()</code>	264
CORBA::UserException	267
<code>CORBA::UserException::UserException()</code>	267
<code>CORBA::UserException::UserException()</code>	268
<code>CORBA::UserException::operator=()</code>	268
<code>CORBA::UserException::_narrow()</code>	268
CORBA::UserOutput	269
<code>CORBA::UserOutput::UserOutput()</code>	270
<code>CORBA::UserOutput::~~UserOutput()</code>	270
<code>CORBA::UserOutput::Output()</code>	270

Part II IDL Interface to the Interface Repository

Common CORBA Data Types	273
<code>CORBA::DefinitionKind</code>	273
<code>CORBA::Identifier</code>	273
<code>CORBA::RepositoryId</code>	274
<code>CORBA::ScopedName</code>	274
CORBA::AliasDef	275
<code>AliasDef::describe()</code>	275
<code>AliasDef::original_type_def</code>	276

CORBA::ArrayDef	277
ArrayDef::element_type	277
ArrayDef::element_type_def	278
ArrayDef::length	278
CORBA::AttributeDef	279
AttributeDef::describe()	279
AttributeDef::mode	280
AttributeDef::type	280
AttributeDef::type_def	281
CORBA::ConstantDef	283
ConstantDef::describe()	283
ConstantDef::type	284
ConstantDef::type_def	284
ConstantDef::value	285
CORBA::Contained	287
Contained::absolute_name()	288
Contained::containing_repository()	288
Contained::defined_in	288
Contained::describe()	289
Contained::id	290
Contained::move ()	290
Contained::name	291
Contained::version	291
CORBA::Container	293
Container::contents()	295
Container::create_alias()	296
Container::create_constant()	296
Container::create_enum()	297
Container::create_exception()	298
Container::create_interface()	299
Container::create_module()	299
Container::create_struct()	300
Container::create_union()	301
Container::describe_contents()	302
Container::lookup()	302
Container::lookup_name()	303

CORBA::EnumDef	305
EnumDef::describe()	305
EnumDef::members	306
CORBA::ExceptionDef	307
ExceptionDef::describe()	307
ExceptionDef::members	308
ExceptionDef::type	309
CORBA::IDLType	311
IDLType::type	311
CORBA::IObject	313
IObject::def_kind	313
IObject::destroy()	313
CORBA::IT_Repository	315
IT_Repository::start()	315
IT_Repository::commit()	316
IT_Repository::rollBack()	316
IT_Repository::active_transactions()	316
CORBA::ModuleDef	317
ModuleDef::describe()	317
CORBA::OperationDef	319
OperationDef::contexts	320
OperationDef::exceptions	320
OperationDef::describe()	320
OperationDef::mode	321
OperationDef::params	321
OperationDef::result	322
OperationDef::result_def	322
CORBA::PrimitiveDef	323
PrimitiveDef::kind	323

CORBA::Repository	325
Repository::create_array()	326
Repository::create_sequence()	326
Repository::create_string()	327
Repository::get_primitive()	327
Repository::describe_contents()	327
Repository::lookup_id()	328
CORBA::SequenceDef	329
SequenceDef::bound	329
SequenceDef::element_type	330
SequenceDef::element_type_def	330
SequenceDef::type	330
CORBA::StringDef	331
StringDef::bound	331
CORBA::StructDef	333
StructDef::describe()	333
StructDef::members	334
CORBA::TypedefDef	335
TypedefDef::describe()	335
CORBA::UnionDef	337
UnionDef::describe()	337
UnionDef::discriminator_type	338
UnionDef::discriminator_type_def	338
UnionDef::members	339

Part III IDL Interface to the Orbix Daemon

IDL Interface to the Orbix Daemon	343
IT_daemon::addDirRights()	347
IT_daemon::addGroupsToServer()	347
IT_daemon::addHostsToGroup()	347
IT_daemon::addHostsToServer()	348
IT_daemon::addInvokeRights()	348
IT_daemon::addInvokeRightsDir()	348
IT_daemon::addLaunchRights()	349
IT_daemon::addLaunchRightsDir()	349
IT_daemon::addMethod()	349
IT_daemon::addSharedMarker()	350
IT_daemon::addUnsharedMarker()	350
IT_daemon::changeOwnerServer()	351
IT_daemon::deleteDirectory()	351
IT_daemon::deleteServer()	351
IT_daemon::delGroupsFromServer()	352
IT_daemon::delHostsFromGroup()	352
IT_daemon::delHostsFromServer()	352
IT_daemon::getServer()	353
IT_daemon::killServer()	353
IT_daemon::LaunchStatus	353
IT_daemon::listActiveServers()	354
IT_daemon::listGroupsInServer()	354
IT_daemon::listHostsInGroup()	354
IT_daemon::listHostsInServer()	355
IT_daemon::listServers()	355
IT_daemon::lookUp()	355
IT_daemon::newDirectory()	356
IT_daemon::newPerMethodServer()	356
IT_daemon::newSharedServer()	357
IT_daemon::newUnSharedServer()	358
IT_daemon::removeDirRights()	358
IT_daemon::removeInvokeRights()	359
IT_daemon::removeInvokeRightsDir()	359
IT_daemon::removeLaunchRights()	359
IT_daemon::removeLaunchRightsDir()	360
IT_daemon::removeMethod()	360
IT_daemon::removeSharedMarker()	360
IT_daemon::removeUnsharedMarker()	361

IT_daemon::serverDetails	361
IT_daemon::serverExists()	362

Appendices

Appendix A

IDL Reference	365
IDL Grammar	365
IDL Grammar: EBNF	366
Keywords	371

Appendix B

System Exceptions	373
-------------------	-----

Index	377
-------	-----

Preface

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

Audience

The *Orbix C++ Programmer's Reference* 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.

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.

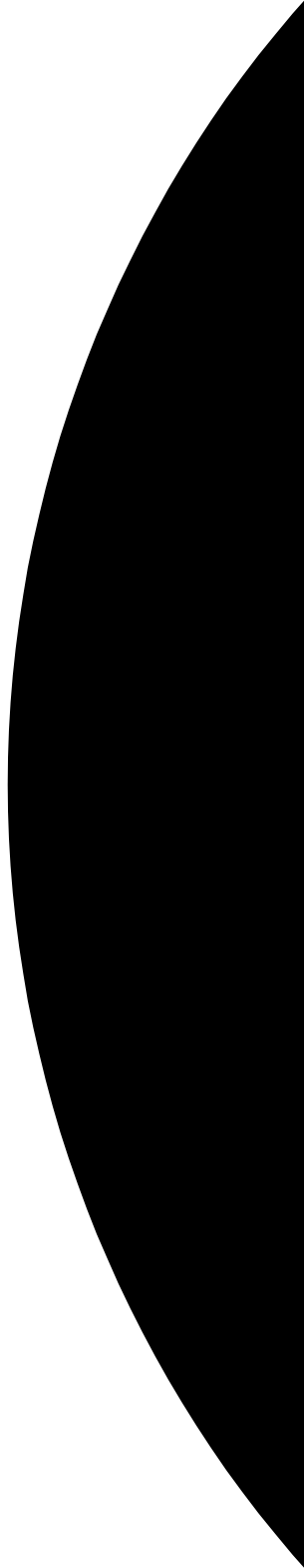
A number sign represents the UNIX command shell prompt for a command that requires root privileges.

> The notation > represents the DOS, Windows NT, or Windows 95 command prompt.

...	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
.	
.	
.	
[]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
	A vertical bar separates items in a list of choices enclosed in { } (braces) in format and syntax descriptions.

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*, *locators*, and *filters*. To assist programmers, each member function in these classes is labelled “CORBA compliant” or “Orbix specific” as appropriate.

Many classes in `CORBA.h` also contain 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.
<code>_ptr</code> return value	The caller is responsible for freeing the object reference or assigning it to a <code>_var</code> 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::_LOCATOR_HOPS

Synopsis `static ULong _LOCATOR_HOPS;`

Description The value used by `_bind()` to determine the number of hosts involved in a search. You can change this if you want to modify how `_bind()` uses `CORBA::locatorClass::lookUp()`.

Notes Orbix specific.

See Also `CORBA::_MAX_LOCATOR_HOPS`
`CORBA::locatorClass`

CORBA::_MAX_LOCATOR_HOPS

Synopsis `static const ULong _MAX_LOCATOR_HOPS;`

Description The absolute maximum number of hops that can be used to fulfil a request; use this to limit the number of hosts involved in a search.

Notes Orbix specific.

See Also `CORBA::_LOCATOR_HOPS`
`CORBA::locatorClass`

CORBA::_OBJECT_TABLE_SIZE_DEFAULT

Synopsis

```
static const 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::default_environment

Synopsis `static Environment default_environment;`

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

The value `default_environment` 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 `default_environment` as the default value for the `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

Orbix 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 insert(  
    const char* _tc_string, void* type);  
static IT_Request_LS 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 Boolean 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 `true` if `obj` is a nil object reference, returns `false` otherwise.

Notes CORBA compliant.

See Also `CORBA::Object::_isNullProxy()`
`CORBA::Object::_isNull()`

CORBA::ORB_init()

Synopsis

```
#ifdef USE_INIT
ORB_ptr ORB_init(int& argc,
                char** argv,
                ORBid orb_identifier,
                CORBA::Environment& env=IT_chooseDefaultEnv());
#endif
```

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`.

To use `ORB_init()`, the macro `USE_INIT` must be `#defined` *before* `CORBA.h` is included.

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 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: -ORB<suffix> <value> where <suffix> is the name of the option being set, and <value> 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: -ORBid Orbix
<code>orb_identifier</code>	A string identifying the ORB. The string "Orbix" identifies the Orbix ORB from IONA Technologies. (Names of ORBs are locally administered by ORB vendors rather than by the OMG.) If this parameter is null, the content of <code>argv</code> 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 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* string_alloc(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* 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 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,
        Environment& IT_env = default_environment);
    void replace(TypeCode_ptr type, void* val,
        Boolean release = 0,
        Environment& IT_env = default_environment);
    Any& operator=(const Any& a);
    ~Any();

    void operator<<=(Short);
    void operator<<=(Long);
    void operator<<=(LongLong);
    void operator<<=(UShort);
```

```
void operator<<=(ULong);
void operator<<=(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>>=(Short&) const;
Boolean operator>>=(Long&) const;
Boolean operator>>=(LongLong) const;
Boolean operator>>=(UShort&) const;
Boolean operator>>=(ULong&) const;
Boolean operator>>=(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;

// 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 20.

```
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 22.

CORBA::Any::Any()

Synopsis `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 `Any(const 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 `Any(TypeCode_ptr type, void* val,
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.
- `val` 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 `~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(TypeCode_ptr, void*, Boolean)`

CORBA::Any::operator=()

Synopsis	<code>Any& operator=(const Any& a);</code>
Description	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.
Notes	CORBA compliant.

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

Synopsis	<pre>void operator<<=(Short); void operator<<=(Long); void operator<<=(LongLong); void operator<<=(UShort); void operator<<=(ULong); void operator<<=(ULongLong); void operator<<=(Float); void operator<<=(Double); void operator<<=(const char*); // Unbounded string. void operator<<=(const Any&); void operator<<=(const TypeCode_ptr&); void operator<<=(Object_ptr); void operator<<=(from_boolean); void operator<<=(from_octet); void operator<<=(from_char);</pre>
-----------------	--

Description Inserts a value of the indicated type into an `Any`.

Any previous value held by the `Any` will be properly deallocated. Each `operator<<=()` takes a copy of the value being inserted (in the case of an object reference, `_duplicate()` is used). The `Any` is then responsible for memory management of the copy.

The insertion function `operator<<=(const char* s)` assumes that the value passed in `s` is an unbounded string. A bounded string must be inserted into an existing `Any` using the function `CORBA::Any::replace()`.

The C++ mapping for IDL types `boolean`, `octet` and `char` cannot be distinguished for the purpose of function overloading. Therefore, the 'helper' types `CORBA::Any::from_boolean`, `CORBA::Any::from_octet`, and `CORBA::Any::from_char` serve to distinguish these types.

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

```
Boolean operator>>=(Short&) const;
Boolean operator>>=(Long&) const;
Boolean operator>>=(LongLong);
Boolean operator>>=(UShort&) const;
Boolean operator>>=(ULong&) const;
Boolean operator>>=(ULongLong);
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;
Boolean operator>>=(to_boolean) const;
Boolean operator>>=(to_octet) const;
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 if the `Any` contains a value of the appropriate type; otherwise they return 0 (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(TypeCode_ptr type, void* val,
            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.

`val` 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 `TypeCode_ptr 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 C++ Programmer's Reference

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 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++
class BOA : public ORB {
public:
    Object_ptr create(const ReferenceData& id,
                     InterfaceDef_ptr intf,
                     ImplementationDef_ptr impl,
                     Environment& env = default_environment);

    ReferenceData* get_id(const Object_ptr,
                          Environment& env = default_environment);

    void dispose(Object_ptr,
                  Environment& env = default_environment) ;

    void change_implementation(Object_ptr obj,
                               ImplementationDef_ptr impl,
                               Environment& env = default_environment);

    Principal_ptr get_principal(Object_ptr obj,
                                Environment& env = default_environment);

    Principal_ptr get_principal(
        Environment& env = default_environment) ;

    Status processNextEvent(
        ULong timeOut=DEFAULT_TIMEOUT,
        Environment& env = default_environment);

    Status processEvents(ULong timeOut = DEFAULT_TIMEOUT,
                         Environment& env = default_environment);

    Boolean isEventPending(
        Environment& env = default_environment) const;

    Boolean anyClientsConnected() const;
    Boolean setNoHangup(Boolean);

#ifdef WANT_ORBIX_FDS
    fd_set getFileDescriptors() const;
#endif
};
```

```
void impl_is_ready(
    ImplementationDef_ptr serverName = "",
    ULong timeOut=DEFAULT_TIMEOUT,
    Environment& env = default_environment);

void impl_is_ready(
    ImplementationDef_ptr serverName,
    Environment& env);

void impl_is_ready(Environment& env);

void deactivate_impl(
    ImplementationDef_ptr,
    Environment &env=default_environment);

void obj_is_ready(
    Object_ptr obj,
    ImplementationDef_ptr impl,
    Environment& env);

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

void deactivate_obj(Object_ptr,
    Environment& env = default_environment);

void continueThreadDispatch(Request&);

const char* myImpRepPath(
    Environment& env = default_environment) const;

const char* myIntRepPath(
    Environment& env = default_environment) const;

const char* myImplementationName(
    Environment& env = default_environment) const;
const char* myMarkerName(
    Environment& env = default_environment) const;
```

```
const char* myMethodName(
    Environment& env = default_environment) const;

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

activationMode myActivationMode(
    Environment& env = default_environment);

Boolean propagateTIEdelete(
    Boolean,
    Environment& env = default_environment);

Boolean filterBadConnectAttempts(
    Boolean,
    Environment& env = default_environment);

Boolean enableLoaders(
    Boolean,
    Environment& env = default_environment);

Filter* getFilter();

static void setImpl(const char* interfaceName,
    DynamicImplementation& rDSISkeleton,
    const char* pMarkerServer = "",
    LoaderClass* pLoader = 0 );
};
```

Notes CORBA compliant.

See Also CORBA::ORB

CORBA::BOA::activationMode

Synopsis

```
enum 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 Boolean anyClientsConnected() const;

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

Return Value Returns true if any clients are connected; returns false otherwise.

Notes Orbix specific.

CORBA::BOA::change_implementation()

Synopsis

```
void change_implementation(Object_ptr obj,
    ImplementationDef_ptr impl,
    Environment& env = default_environment);
```

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 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

```
Object_ptr create(  
    const ReferenceData& id,  
    InterfaceDef_ptr intf,  
    ImplementationDef_ptr impl,  
    Environment& env = default_environment);
```

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 deactivate_impl(ImplementationDef_ptr impl,  
                    Environment& env = default_environment);
```

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 deactivate_obj(Object_ptr obj,  
    Environment& env = default_environment);
```

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 dispose(Object_ptr obj,  
    Environment& env = default_environment) 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()

- Synopsis** `Boolean enableLoaders(Boolean value,
Environment& env = default_environment);`
- Description** 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 *if* it exists, it may be advisable to avoid involving the loaders if the object cannot be found.
- By default, loaders are enabled.
- Parameters**
- `value` A true value enables loaders; false disables loaders.
- Return Value** Returns the previous setting.
- Notes** Orbix specific.
- See Also** `CORBA::LoaderClass`

CORBA::BOA::filterBadConnectAttempts()

- Synopsis** `Boolean filterBadConnectAttempts(Boolean value,
Environment& env = default_environment);`
- Description** By default, an exception is raised 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. Such bad connection attempts can be caused, for example, by a connection attempt from an old version of Orbix (version 1.2 or earlier).
- If you want to have Orbix silently handle such attempts without raising an exception to `CORBA::BOA::impl_is_ready()`, `CORBA::BOA::obj_is_ready()`, or `CORBA::BOA::processEvents()`, you can call `filterBadConnectAttempts()` passing true for the parameter value.
- Return Value** Returns the previous setting. The default is disabled (that is, by default, bad connection attempts cause a `CORBA::COMM_FAILURE` exception to be raised).
- Notes** Orbix specific.

CORBA::BOA::get_id()

Synopsis

```
ReferenceData* get_id(Object_ptr obj,  
    Environment& env = default_environment);
```

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::get_principal()

Synopsis

```
Principal_ptr get_principal(  
    Environment& env = default_environment);  
Principal_ptr get_principal(Object_ptr ignored,  
    Environment& env = default_environment);
```

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.

See Also `CORBA::Principal`

CORBA::BOA::getFileDescriptors()

Synopsis

```
#include <sys/types.h>
#ifdef WANT_ORBIX_FDS
fd_set getFileDescriptors() const;
#endif
```

Description Gets the set of file descriptors scanned by Orbix to detect incoming events. When using libraries or systems that depend on the UNIX `select()` system call you may need to know which file descriptors are scanned by Orbix.

Note: This function is defined only if the following preprocessor directive is issued in the C++ file before including `CORBA.h`:

```
#define WANT_ORBIX_FDS
```

Return Value Returns a set of file descriptors.

Notes Orbix specific.

See Also `CORBA::Object::_fd()`

CORBA::BOA::getFilter()

Synopsis

```
Filter_ptr 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 impl_is_ready(  
    Environment& env = default_environment);  
void impl_is_ready(ImplementationDef_ptr serverName,  
    Environment& env = default_environment);  
void impl_is_ready(  
    ImplementationDef_ptr serverName = "",  
    ULong timeOut = DEFAULT_TIMEOUT,  
    Environment& env = default_environment);
```

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, or using the locator, 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::isEventPending()

Synopsis `Boolean isEventPending(
Environment& env = default_environment) 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 `true` if there is a pending event, returns `false` otherwise.

Notes Orbix specific.

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

CORBA::BOA::myActivationMode()

- Synopsis** `activationMode myActivationMode(
 Environment& env = default_environment);`
- 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* myImplementationName(
 Environment& env = default_environment) 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* myImpRepPath(
 Environment& env = default_environment) 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* myIntRepPath(
 Environment& env = default_environment) 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* myMarkerName(
 Environment& env = default_environment) 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* myMarkerPattern(
 Environment& env = default_environment);`
- 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* myMethodName(  
    Environment& env = default_environment) 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 obj_is_ready(Object_ptr obj,  
    ImplementationDef_ptr impl,  
    Environment& env = default_environment);  
void obj_is_ready(Object_ptr obj,  
    ImplementationDef_ptr impl,  
    ULong timeOut = DEFAULT_TIMEOUT,  
    Environment& env = default_environment);
```

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 processEvents(ULong timeOut = DEFAULT_TIMEOUT,
Environment& env = default_environment);`

- 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 processNextEvent(ULong timeOut = DEFAULT_TIMEOUT,
Environment& env = default_environment);
```

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 Boolean propagateTIEdelete(Boolean value,
Environment& env = default_environment);

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::setImpl()

Synopsis

```
static void setImpl(const char* InterfaceName,  
                  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

```
Boolean setNoHangup(Boolean value);
```

Description

By default, the `CORBA::BOA::impl_is_ready()` and `CORBA::BOA::obj_is_ready()` functions time out when a period (user-defined or defaulted) has elapsed between events. An event is an incoming operation call or the connection or disconnection by a client.

This means that `impl_is_ready()` and `obj_is_ready()` can time out when clients are idle for a period. If a server is to remain active while it has any clients, active or not, it can call the function `setNoHangup(true)` on the `CORBA::Orbix` object.

Parameters

`value` When `setNoHangup(true)` is called, the time-out period to `impl_is_ready()` and `obj_is_ready()` specifies the amount of time a server remains waiting while there are no client connections to it. When `setNoHangup(false)` is called, the time-out period to `impl_is_ready()` and `obj_is_ready()` specifies the amount of time a server remains waiting while there are no events (operation calls, connections, disconnections).

When set to `true`, `impl_is_ready()` and `obj_is_ready()` do not time out while there are clients connected to the server; when set to `false`, `impl_is_ready()` and `obj_is_ready()` can time out if the specified period elapses between events. The default is `false`.

Return Value Returns the previous setting.

Notes Orbix specific.

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

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 passed to an operation also includes the identifiers of its parent(s). The called function can decide whether to use just the context actually passed, or the hierarchy above it.

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,
                        Environment& env = default_environment);

    Status set_values(NVList_ptr values,
                    Environment& env = default_environment);

    Status get_values(const char* start_scope,
                    Flags op_flags,
                    const char* prop_name,
                    NVList_ptr& values,
                    Environment& env = default_environment) ;

    Status delete_values(const char* prop_name,
                       Environment& env = default_environment);

    Status create_child(const char* ctx_name,
                      Context_ptr& child_ctx,
                      Environment& env = default_environment);

    const char* context_name(
        Environment& env = default_environment) const;

    Context_ptr parent(
        Environment& env = default_environment) const;

    ORBStatus _delete (const CORBA::Flags &del_flags,
                     Environment& env = default_environment);

    ULong get_count(
        Environment& env = default_environment) const;

    ULong get_count_all(
        Environment& env = default_environment) 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,
    Environment& env = default_environment);

static Context_ptr __stdcall IT_create(
    const char* name,
    Context_ptr parent = 0,
    Environment& env = default_environment);

static Context_ptr __stdcall IT_create(
    Request& IT_r,
    Environment& env = default_environment);

static Context_ptr __stdcall _duplicate(
    Context_ptr obj,
    Environment& env = default_environment);

static Context_ptr __stdcall _nil(
    Environment& env = default_environment);
};
```

Notes

CORBA compliant.

See Also

CORBA::ContextIterator
CORBA::NVList
CORBA::Flags

CORBA::Context::Context()

Synopsis

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

Description

Creates a new context (possibly a child context).

Parameters

`parent` The parent context, if any.

Notes

Orbis 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

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

Description

Creates a new context (possibly a child context).

Parameters

`name` The name of the context.

`parent` The parent context, if any.

Notes

Orbis 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** `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** `~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 Context_ptr _duplicate(
    Context_ptr obj,
    Environment& env = default_environment);
```
- 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** `static Context_ptr _nil(
 Environment& env = default_environment);`
- Description** Returns a nil object reference for a Context object.
- Notes** CORBA compliant.
- See Also** `CORBA::is_nil()`

CORBA::Context::context_name()

- Synopsis** `const char* context_name(
 Environment& env=default_environment) const;`
- Description** Returns the name of the Context object.
- See Also** `CORBA::Context::create_child()`

CORBA::Context::create_child()

- Synopsis** `Status create_child(const char* ctx_name,
 Context_ptr& child_ctx,
 Environment& env = default_environment);`
- Description** Creates a child context of the current context. When a child context is passed as a parameter to an operation, any searches (using `CORBA::Context::get_values()`) looks in parent contexts if necessary to find matching property names.
- Parameters**
- | | |
|------------------------|---|
| <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** `CORBA::Context::get_values()`

CORBA::Context::delete_values()

- Synopsis** `Status delete_values(const char* prop_name,
 Environment& env = default_environment);`
- 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** `Long get_count(
 Environment& env = default_environment) const;`
- Description** Finds the number of property/value pairs in the context.
- Notes** Orbix specific.
- See Also** `CORBA::Context::get_count_all()`

CORBA::Context::get_count_all()

- Synopsis** `Long get_count_all(
 Environment& env = default_environment) const;`
- Description** Finds the number of property/value pairs in this context and all its parent contexts.
- Notes** Orbix specific.
- See Also** `CORBA::Context::get_count()`

CORBA::Context::get_values()

Synopsis

```
Status get_values(  
    const char* start_scope,  
    const Flags op_flags,  
    const char* prop_name,  
    NVList_ptr& values,  
    Environment& env = default_environment);
```

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 Context_ptr IT_create(  
    Context_ptr parent=0,  
    Environment& env = default_environment);  
static Context_ptr IT_create(  
    const char* name,  
    Context_ptr parent=0,  
    Environment& env = default_environment);  
static Context_ptr IT_create(  
    Request& IT_r,  
    Environment& env = default_environment);
```

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

```
Context_ptr parent(  
    Environment& env = default_environment) const;
```

Description

Returns the parent of the `Context` object.

See Also

`CORBA::Context::create_child()`

CORBA::Context::set_one_value()

Synopsis

```
Status set_one_value(const char* prop_name,  
                    const Any& value,  
                    Environment& env = default_environment);
```

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

`prop_name` The name of the property to add.
`value` 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 set_values(const NVList_ptr values,  
                 Environment& env = default_environment);
```

Description

Sets one or more property values in the context. The previous value of the property, if any, is discarded.

Parameters

`values` An `NVList` containing the `property_name:values` to add or change. In the `NVList`, the `flags` field must be set to zero, and the `TypeCode` associated with an attribute value must be `CORBA::_tc_string`.

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 ContextIterator(const Context* context);

Description Constructor. Creates an iterator for context.

Notes Orbix specific.

CORBA::ContextIterator::~~ContextIterator()

Synopsis ~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

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,
                       Environment& IT_env = default_environment) = 0;
protected:
    DynamicImplementation();
    virtual ~DynamicImplementation();
};
```

Notes CORBA compliant.

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

CORBA::DynamicImplementation::DynamicImplementation()

Synopsis `DynamicImplementation();`

Description Default constructor.

Notes CORBA compliant.

CORBA::DynamicImplementation:: ~DynamicImplementation()

Synopsis `~DynamicImplementation();`

Description Destructor.

Notes CORBA compliant.

CORBA::DynamicImplementation::invoke()

Synopsis `virtual void invoke(ServerRequest& request,
 Environment& env,
 Environment& IT_env = default_environment) = 0;`

Description The `invoke()` function is informed of incoming operation and attribute requests to a server. An implementation of `invoke()` (in a derived class of `CORBA::DynamicImplementation`) is known as a Dynamic Implementation Routine (DIR).

Parameters

`request` 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 `DynamicImplementation` has been registered to handle the target interface.

`env` Contains the environment associated with the `request` parameter.

`IT_env` May be used to return exception information when C++ exceptions are not supported by the compiler.

Notes CORBA compliant.

See Also `CORBA::ServerRequest`

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. Where the C++ host compiler does not support C++ exception handling, you use it primarily transmit exceptions back to a caller via the default `CORBA::Environment` parameter. How you can use this parameter for exception handling is described in the *Orbix C++ Programmer's Guide*. 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). Where appropriate, the function descriptions below assume a non-exception handling compiler.

Whether or not the compiler supports exceptions, you can use this default parameter, in Orbix, to set a time-out value for remote calls and to pass security information.

CORBA

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

Orbix

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

    Request_ptr m_request;

    Environment(Exception*);
    Environment(Request&);
    Environment();

    ~Environment();
    Environment(const Environment&);

    const Environment& operator=(const Environment&);

    Environment(SystemException*);
    const Environment& operator=(Exception*);
```

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

operator int() const;

void exception(Exception* e);
Exception* exception() const;
void clear();

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

void propagate(const Exception*);
void propagate(const SystemException*);

friend ostream& operator<< (ostream& o, Environment&);

static Environment_ptr IT_create(Exception*,
    Environment& env = default_environment);
static Environment_ptr IT_create(Request&,
    Environment& env = default_environment);
static Environment_ptr IT_create(
    Environment& env = default_environment);
static Environment_ptr IT_create(const Environment&,
    Environment& env = default_environment);
static Environment_ptr IT_create(SystemException*,
    Environment& env = default_environment);
static Environment_ptr _duplicate(
    Environment_ptr obj,
    Environment& env = default_environment);
static Environment_ptr _nil(
    Environment& env = default_environment);
};
```

Notes

CORBA compliant.

See Also

CORBA::default_environment
CORBA::BOA::get_principal()

CORBA::Environment::Environment()

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

CORBA::Environment::Environment()

- Synopsis** `Environment(const Environment& env);`
- Description** Copy constructor.
- Notes** Orbix specific.
- See Also** `CORBA::ORB::create_environment()`
`CORBA::Environment::IT_create()`
Other `Environment` constructors.

CORBA::Environment::Environment()

- Synopsis** `Environment(SystemException* se);`
- Description** Conversion from a `SystemException`. Constructs an `Environment` that contains the exception denoted by `se`.
- Notes** Orbix specific.
- See Also** `CORBA::SystemException`
`CORBA::ORB::create_environment()`
`CORBA::Environment::IT_create()`
Other `Environment` constructors.

CORBA::Environment::~~Environment()

- Synopsis** `~Environment();`
- Description** Destructor.
- Notes** Orbix specific.

CORBA::Environment::operator=()

- Synopsis** `const Environment& operator=(const Environment& env);`
- Description** Assignment operator.
- Notes** Orbix specific.
- See Also** `Environment::operator=(CORBA::Exception* e)`
`Environment::operator=(CORBA::SystemException* se)`

CORBA::Environment::operator=()

- Synopsis** `const Environment& operator=(Exception* e);`
- Description** Assignment from an Exception.
- Notes** Orbix specific.
- See Also** `Environment::operator=(const Environment& env)`
`Environment::operator=(CORBA::SystemException* se)`

CORBA::Environment::operator=()

- Synopsis** `const Environment& operator=(SystemException* se);`
- Description** Assignment from a system exception.
- Notes** Orbix specific.
- See Also** `Environment::operator=(const Environment& env)`
`Environment::operator=(CORBA::Exception* e)`

operator<<()

- Synopsis** `friend ostream& operator<<(ostream& o, Environment env);`
- Description** Overloads `operator<<()` to output exception information contained in the parameter `env` on ostream `o`.
- Notes** Orbix specific.

CORBA::Environment::_duplicate()

- Synopsis** `static Environment_ptr _duplicate(Environment_ptr obj, Environment& env = default_environment);`
- Description** Increments the reference count of `obj`.
- Return Value** Returns a reference to itself.

Notes CORBA compliant.

See Also `CORBA::release()`

CORBA::Environment::_nil()

Synopsis

```
static Environment_ptr _nil(  
    Environment& env = default_environment);
```

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

Notes CORBA compliant.

See Also `CORBA::is_nil()`

CORBA::Environment::clear()

Synopsis

```
void 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::exception()

Synopsis `Exception* exception() const;`

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 `Environment` retains ownership of the `Exception` returned. Thus, once the `Environment` is destroyed, or its `Exception` cleared, the reference is no longer valid.

Notes CORBA compliant.

See Also `CORBA::Environment::exception()`
`CORBA::Environment::exception(CORBA::Exception* e)`
`CORBA::Environment::clear()`

CORBA::Environment::exception()

Synopsis `void exception(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 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 IT_create(  
    Exception*, Environment& env = default_environment);  
static Environment_ptr IT_create(  
    Request&, Environment& env = default_environment);  
static Environment_ptr IT_create(  
    Environment& env = default_environment);  
static Environment_ptr IT_create(  
    const Environment&, Environment& env = default_environment);  
static Environment_ptr IT_create(  
    SystemException*, Environment& env = default_environment);
```

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 initialise 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::m_request

Synopsis	<code>Request* m_request;</code>
Description	A pointer to the <code>Environment</code> 's associated <code>Request</code> object, if any.
Notes	Orbix specific.
See Also	<code>CORBA::Request</code>

CORBA::Environment::propagate()

Synopsis	<code>void propagate(const Exception* e);</code> <code>void propagate(const SystemException* e);</code>
Description	If an operation implementation receives an exception when it calls another operation, it can propagate that exception back to its own caller. Both system and user-defined exceptions can be propagated in this way via the <code>Environment</code> parameter.

As an example, the code below assumes that interface `Bank` is implemented by class `Bank_i`, and that the `newAccount()` function calls another function (using the reference `pPtr`, whose nature is not of concern here) that can raise a `Bank::Reject` exception:

```
Account_ptr Bank_i::newAccount(const char* name,
    CORBA::Environment& pe =
        CORBA::default_environment) {
    someType_ptr pPtr = ....;
    TRY {
        pPtr->op(.....,IT_X);
    }
    CATCH (Bank::Reject, rej) {
        pe.propagate(rej);
    }
    ENDRY
}
```

Notes	Orbix specific.
--------------	-----------------

CORBA::Environment::timeout()

- Synopsis** `ULong timeout() const;`
- Description** Gets the timeout, in milliseconds, for remote calls for this `Environment`.
- Notes** Orbix specific.
- See Also** `CORBA::Environment::timeout(CORBA::ULong t)`
`CORBA::ORB::defaultTxTimeout()`

CORBA::Environment::timeout()

- Synopsis** `void timeout(ULong t);`
- Description** 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 `CORBA::ORB::defaultTxTimeout()`.
- This function is effective once a connection has been established between the client and server.
- Parameters**
- `t` The timeout value in milliseconds.
- Exceptions** If a reply is not received within the given timeout interval, an invocation using this `Environment` value fails with a `CORBA::COMM_FAILURE` exception.
- Notes** Orbix specific.
- See Also** `CORBA::Environment::timeout()`
`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 Exception();

Description Default constructor.

Notes CORBA compliant.

See Also Other Exception constructors.

CORBA::Exception::Exception()

Synopsis Exception(const Exception& e);

Description Copy constructor.

Notes CORBA compliant.

See Also Other Exception constructors.

CORBA::Exception::~~Exception()

Synopsis

```
virtual ~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 Exception& operator=(  
    const 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 C++ 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 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 ~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 `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 `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 ~ExtraRegistryCVHandler();
Description Destructor.
Notes Orbix specific.

CORBA::ExtraRegistryCVHandler::GetRegKey()

Synopsis HKEY 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 C++ Programmer's 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 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 ~Filter();

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

Notes Orbix specific.

CORBA::Filter::inReplyFailure()

Synopsis virtual void inReplyFailure(Request& r, 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 Boolean inReplyPostMarshal(
Request& r, 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 Boolean inReplyPreMarshal(
 Request& r, 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 Boolean inRequestPostMarshal(
 Request& r, 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 inRequestPreMarshal(  
    Request& r, 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 (Orbix-MT only) 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 outReplyFailure(Request& r, 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 Boolean outReplyPostMarshal(
 Request& r, 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 Boolean outReplyPreMarshal(
 Request& r, 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 Boolean outRequestPostMarshal(
    Request& r, 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 Boolean outRequestPreMarshal(
    Request& r, 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 `Flags();`

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

Notes Orbix specific.

See Also Other `Flags` constructors.

CORBA::Flags::Flags()

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

CORBA::Flags::isNil()

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

CORBA::Flags::reset()

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

CORBA::Flags::setArgDef()

- Synopsis** `void setArgDef(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 setf(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,
 Environment& env = default_environment);

 static ImplementationDef_ptr _duplicate (
 ImplementationDef_ptr obj,
 Environment& env = default_environment);

 static ImplementationDef_ptr _nil (
 Environment& env = default_environment);
};

Notes CORBA compliant.

CORBA::ImplementationDef::_duplicate()

Synopsis static ImplementationDef_ptr _duplicate(
 ImplementationDef_ptr obj,
 Environment &env = default_environment);

Description Increments the reference count of obj.

Notes CORBA compliant.

See Also CORBA::release()

CORBA::ImplementationDef::_nil()

- Synopsis** `static ImplementationDef_ptr _nil(
 Environment &env = default_environment);`
- Description** Returns a nil object reference for `ImplementationDef`.
- Notes** CORBA compliant.
- See Also** `CORBA::is_nil()`

CORBA::ImplementationDef::IT_create()

- Synopsis** `static ImplementationDef_ptr IT_create(
 const char* impl,
 Environment& env = default_environment);`
- Description** In the absence of a CORBA-specified way to create an `ImplementationDef` pseudo object in the current standard C++ mapping, Orbix provides the `IT_create()` function to initialise an object reference for an `ImplementationDef`.
- Use of this function is recommended in preference to C++ operator `new` to ensure memory management consistency.
- Notes** 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++
class IT_IOCallback {
public:
    virtual ~IT_IOCallback() {}

    virtual void OrbixFDOpen(int fd);
    virtual void OrbixFDClose(int fd);

    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::ForeignFDExcept()

Synopsis

```
virtual void ForeignFDExcept(int fd);
```

Description

Defines the action to be taken when an exception event occurs on a foreign file descriptor registered with the Orbix event loop.

Parameters

`fd` The foreign file descriptor on which an event occurred.

Notes

Orbix specific.

CORBA::IT_IOCallback::ForeignFDRead()

Synopsis

```
virtual void ForeignFDRead(int fd);
```

Description

Defines the action to be taken when a read event occurs on a foreign file descriptor registered with the Orbix event loop.

Parameters

`fd` The foreign file descriptor on which an event occurred.

Notes

Orbix specific.

CORBA::IT_IOCallback::ForeignFDWrite()

Synopsis

```
virtual void ForeignFDWrite(int fd);
```

Description

Defines the action to be taken when a write event occurs on a foreign file descriptor registered with the Orbix event loop.

Parameters

`fd` The foreign file descriptor on which an event occurred.

Notes

Orbix specific.

CORBA::IT_IOCallback::OrbixFDClose()

Synopsis

```
virtual void 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 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_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 C++ Programmer's 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*,
        Environment &env = IT_chooseDefaultEnv());
};
```

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

IT_reqTransformer* getMyReqTransformer();
};
```

Notes Orbix specific.

See Also CORBA::Request

CORBA::IT_reqTransformer::free_buf()

Synopsis virtual void free_buf(unsigned char* data,
 ULong actual_sz,
 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 Boolean transform(  
    unsigned char*& data,  
    ULong& actual_sz,  
    ULong& allocd_sz,  
    Boolean send,  
    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 (false) 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* 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 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**
- | | |
|------------------------|-----------------------|
| <code>host_name</code> | The name of the host. |
|------------------------|-----------------------|
- Notes** Orbix specific.

CORBA::ORB::getMyReqTransformer()

- Synopsis** `IT_reqTransformer* getMyReqTransformer();`
- Description** Returns the registered transformer object; returns 0 if no transformer registered.
- Notes** Orbix specific.
- See Also** `CORBA::setMyReqTransformer()`

CORBA::ORB::setMyReqTransformer()

- Synopsis** `IT_reqTransformer* setMyReqTransformer(
IT_reqTransformer*,
Environment &env = IT_chooseDefaultEnv());`
- Description** Registers an `IT_reqTransformer` object as the default transformation for Requests leaving or entering the address space.
- Parameters**
- | | |
|--------------------------|--------------------------------------|
| <code>transformer</code> | 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 setReqTransformer(IT_reqTransformer* transformer,  
    const char* server,  
    const char* host = 0,  
    Environment& env = 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

<code>transformer</code>	A pointer to the transformer to be registered.
<code>server</code>	The name of the server for which the transformer is to be used.
<code>host</code>	The name of the host.

Notes

Orbix specific.

See Also

CORBA::setMyReqTransformer()

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 LoaderClass(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 ~LoaderClass();

Description The destructor.

Notes Orbix specific.

CORBA::LoaderClass::load()

Synopsis

```
virtual Object_ptr load(const char* interface,
    const char* marker, 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 record(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 Boolean rename(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 save(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::locatorClass

Synopsis Class CORBA::locatorClass is an abstract base class that defines the interface to a location server. Orbix uses the installed locator to find a target object in the distributed system when `_bind()` is called with a null host name.

You may override the default locator by defining and implementing a derived class of CORBA::locatorClass as described in the *Orbix C++ Programmer's Guide*.

Orbix

```
// C++
class locatorClass {
public:
    virtual CORBA::ORB::IT_StringSeq lookUp(
        const char* ServiceName,
        ULong MaxHops,
        const Context& context,
        Environment&) = 0;
};
static locatorClass* locator;
```

Notes Orbix specific.

CORBA::locatorClass::locatorClass()

Synopsis `locatorClass();`

Description Default constructor.

Notes Orbix specific.

CORBA::locatorClass::lookUp()

Synopsis

```
virtual CORBA::ORB::IT_StringSeq lookUp(const char* ServiceName,  
    ULong MaxHops, const Context& context);
```

Description

Searches for a server. It is called on the locator pointed to by `CORBA::locator`, when `_bind()` is called with a null host name. Any parameters to `_bind()` are passed to `lookUp()`.

Parameters

<code>ServiceName</code>	The name of the server being sought.
<code>MaxHops</code>	In the default locator, this is interpreted as the maximum number of machines to search for the required server. An interpretation similar to this one should be retained in a user-defined locator if it is to be used without changing client code that explicitly calls <code>lookUp()</code> .
<code>context</code>	This allows a client to pass extra information to the locator: for example, constraints on how to search for the server. You can use the context parameter to define properties to be used when deciding between a set of servers with the same name. The <code>Context</code> passed to <code>lookUp()</code> originates in the <code>Context</code> value passed by the application programmer to an <code>_bind()</code> call. The default locator ignores this parameter.

Return Value

The default locator returns a list of names of hosts on which that server is registered in the Implementation Repository. The default implementation of the locator randomises the sequence before returning it. This is a basic technique in load balancing to avoid swamping any one server. An empty sequence is returned if no host names can be found for the specified server.

Notes

Orbix specific.

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(
        Environment& env = default_environment);
    static NamedValue_ptr IT_create(const NamedValue&,
        Environment& env = default_environment);

    static NamedValue_ptr _duplicate(
        NamedValue_ptr obj,
        Environment& env = default_environment);
```

```
static NamedValue_ptr _nil(  
    Environment& env = default_environment);  
};
```

Notes CORBA compliant.

See Also CORBA::NVList
CORBA::Request
CORBA::Flags

CORBA::NamedValue::NamedValue()

Synopsis 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 NamedValue(const 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 `~NamedValue();`

Description Destructor.

Notes Orbix specific.

See Also `CORBA::release()`

CORBA::NamedValue::operator=()

Synopsis `const NamedValue& operator=(
 const NamedValue& nv);`

Description Assignment operator.

Notes Orbix specific.

CORBA::NamedValue::_duplicate()

Synopsis `static NamedValue_ptr _duplicate(
 NamedValue_ptr obj,
 Environment& env = default_environment);`

Description Increments the reference count of obj.

Return Value Returns a reference to self.

Notes CORBA compliant.

See Also `CORBA::release()`

CORBA::NamedValue::_nil()

Synopsis `static NamedValue_ptr _nil(
 Environment& env = default_environment);`

Description Returns a nil object reference for a `NamedValue`.

Notes CORBA compliant.

See Also `CORBA::is_nil()`

CORBA::NamedValue::flags()

Synopsis `Flags flags() const;`

Description Returns the `CORBA::Flags` object associated with the `NamedValue`.

Notes CORBA compliant.

See Also `CORBA::Flags`

CORBA::NamedValue::IT_create()

Synopsis

```
static NamedValue_ptr IT_create(  
    Environment& env = default_environment);  
static NamedValue_ptr IT_create(const NamedValue&,   
    Environment& env = default_environment);
```

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::name()

- Synopsis** `const char* 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::value()

- Synopsis** `Any* 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 NullLoaderClass();

Description Default constructor.

Notes Orbix specific.

See Also CORBA::LoaderClass::LoaderClass()

CORBA::NullLoaderClass::record()

Synopsis

```
virtual void record(Object_ptr obj_ref obj,  
                   char*& marker, 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(Environment& env = default_environment) const;

    NamedValue_ptr add(
        Flags item_flags,
        Environment& env = default_environment);

    NamedValue_ptr add_item(
        const char* item_name,
        Flags item_flags
        Environment& env = default_environment);

    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,
    Environment& env = default_environment);

NamedValue_ptr item(
    CORBA::ULong index,
    Environment& env = default_environment);

Status remove(CORBA::Long l,
    Environment& env = default_environment);

NVList();

NVList(Long l, Environment& env = default_environment);

NVList(const NVList&);

const NVList& operator=(const NVList&);

~NVList();

static NVList_ptr IT_create(
    Environment& env = default_environment);

static NVList_ptr IT_create(
    Long,
    Environment& env = default_environment);

static NVList_ptr IT_create(
    const NVList&,
    Environment& env = default_environment);

static NVList_ptr _duplicate(
    NVList_ptr obj,
    Environment& env = default_environment);

static NVList_ptr _nil(
    Environment& env = default_environment);
};
```

Notes CORBA compliant.

See Also CORBA::NamedValue
CORBA::Request

CORBA::NVList::NVList()

Synopsis 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 NVList(Long size,
Environment& env = default_environment);

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()

Synopsis NVList(const NVList&);

Description Copy constructor.
Notes Orbix specific.
See Also Other `NVList` constructors.

CORBA::NVList::~~NVList()

Synopsis `~NVList();`
Description Destructor.
Notes Orbix specific.

CORBA::NVList::operator=()

Synopsis `const NVList& operator=(const NVList&);`
Description Assignment operator.
Notes Orbix specific.

CORBA::NVList::_duplicate()

Synopsis `static NVList_ptr _duplicate(
 NVList_ptr obj,
 Environment& env = default_environment);`
Description Increments the reference count of `obj`.
Return Value Returns a reference to self.
Notes CORBA compliant.
See Also `CORBA::release()`

CORBA::NVList::_nil()

Synopsis `static NVList_ptr _nil(
 Environment& env = default_environment);`

Description Returns a nil object reference for an NVList object.

Notes CORBA compliant.

See Also CORBA::is_nil()

CORBA::NVList::add()

Synopsis

```
NamedValue_ptr add(Flags item_flags,  
Environment& env = default_environment);
```

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

```
NamedValue_ptr add_item(const char* item_name,  
Flags item_flags,  
Environment& env = default_environment);
```

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

```
NamedValue_ptr add_value(const char* item_name,  
    const Any& value, Flags item_flags,  
    Environment& env = default_environment);
```

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

```
NamedValue_ptr add_item_consume(char* item_name,  
    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

```
NamedValue_ptr add_value_consume(char* item_name,  
    Any* item_value,  
    Flags item_flags);
```

Description

Creates a `NamedValue` with `name`, `value` 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

```
ULong count(  
    Environment& env = default_environment) const;
```

Description

Returns the number of elements in the `NVList`.

Notes

CORBA compliant.

CORBA::NVList::IT_create()

Synopsis

```
static NVList_ptr IT_create(  
    Environment& env = default_environment);  
static NVList_ptr IT_create(ULong,  
    Environment& env = default_environment);  
static NVList_ptr IT_create(const NVList&
```

```
Environment& env = default_environment);
```

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

```
NamedValue_ptr item(ULong index,  
Environment& env = default_environment);
```

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

```
Status remove(ULong index,  
Environment& env = default_environment);
```

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.

Orbix

```
// 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 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** `NVListIterator(NVList_ptr l);`
- Description** Constructor. Creates an iterator for `NVList l`.
- Notes** Orbix specific.
- See Also** Other `NVListIterator` constructor.

CORBA::NVListIterator::operator()

- Synopsis** `NamedValue_ptr 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** `setList(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;

class Object {
public:
    ImplementationDef_ptr _get_implementation(
        Environment& env = default_environment) ;
```

```
InterfaceDef_ptr _get_interface(
    Environment& env = default_environment);

Request_ptr _request(
    Identifier operation,
    Environment& env = default_environment);

Status _create_request(
    Context_ptr ctx,
    const char* operation,
    NVList_ptr arg_list,
    NamedValue_ptr& result,
    Request_ptr& request,
    Flags req_flags,
    Environment& env = default_environment);

Boolean _is_a(const char* logical_type_id,
    Environment &env = default_environment);

Boolean _non_existent(Environment &env =
    default_environment);

Boolean _is_equivalent(const Object_ptr obj,
    Environment& env = default_environment);

ULong _hash(ULong maximum,
    Environment& env=IT_chooseDefaultEnv );

Object(Boolean normal = false);
Object(const Object&);
Object(const Object_ptr);

const Object& operator=(const Object&);

virtual ~Object();

Object(const char*);

Object(const char* host, const char* impl,
    const char* marker, const char* intfHost,
    const char* intfImpl,
    const char* intfMarker);
```

```
Identifier _object_to_string(
    Environment& env = default_environment);

const char* _host(
    Environment& env = default_environment) const;

const char* _implementation(
    Environment& env = default_environment) const;

const char* _marker(
    Environment& env = default_environment) const;
Boolean _marker(const char*,
    Environment& env = default_environment);

const char* _interfaceHost(
    Environment& env = default_environment) const;

const char* _interfaceImplementation(
    Environment& env = default_environment) const;

const char* _interfaceMarker(
    Environment& env = default_environment) const;

Boolean _isNullProxy(
    Environment& env = default_environment) const;

Boolean _isNull(
    Environment& env = default_environment) const;

void* _attachPre(void* b);
void* _attachPost(void* b);
void* _getPre() const;
void* _getPost() const;
virtual void* _deref();

#ifdef _REENTRANT
    Boolean _enableInternalLock(Boolean,
        Environment& env = default_environment);
#endif

#ifdef WANT_ORBIX_FDS
    int _fd (Environment& env = default_environment) const;
#endif
```

```
virtual LoaderClass* _loader(  
    Environment& env = default_environment) const;  
  
virtual void _save(  
    Environment& env = default_environment);  
  
virtual Boolean _isRemote(  
    Environment& env = default_environment) const;  
  
void _closeChannel(Environment& env =  
    default_environment);  
  
Boolean _hasValidOpenChannel(Environment& env =  
    default_environment);  
  
ULong _refCount(  
    Environment& env = default_environment) const;  
  
static Object_ptr _duplicate(  
    Object_ptr obj,  
    Environment& env = default_environment);  
static Object_ptr _nil(  
    Environment& env = default_environment);  
};
```

Notes CORBA compliant.

CORBA::Object::Object()

Synopsis `Object(Boolean normal = false);`

Description Default constructor. If the parameter `normal` is set to `true` a null proxy is created.

Notes Orbix specific. You should not normally need to use this function.

See Also Other `Object` constructors.

CORBA::Object::Object()

Synopsis	<code>Object(const Object& obj);</code>
Description	Copy constructor.
Notes	Orbix specific.
See Also	Other <code>Object</code> constructors.

CORBA::Object::Object()

Synopsis	<code>Object(const Object_ptr obj_ptr);</code>
Description	Conversion from an <code>Object_ptr</code> .
Notes	Orbix specific.
See Also	Other <code>Object</code> constructors.

CORBA::Object::Object()

Synopsis	<code>Object(const char* obj_ref_string);</code>
Description	<p>Conversion from a stringified object reference. An Orbix stringified object reference has the form:</p> <pre>:\host:serverName:marker:IFR_host:IFR_Server: interfaceMarker</pre> <p>Refer to <code>CORBA::ORB::object_to_string()</code> for an explanation of these fields.</p> <p>This constructor is normally used with the DII where it is not necessary to link in the generated code for the IDL C++ class.</p> <p>If the string <code>obj_ref_string</code> is not a valid stringified object reference, Orbix creates a null object.</p>
Notes	Orbix specific. The CORBA compliant version of this function is <code>CORBA::ORB::string_to_object()</code> .
See Also	<p><code>CORBA::BOA::create()</code> <code>CORBA::ORB::string_to_object()</code> <code>CORBA::Object::_isNull()</code></p> <p>Other <code>Object</code> constructors.</p>

CORBA::Object::Object()

Synopsis

```
Object(const char* host,
        const char* serverName,
        const char* marker,
        const char* IFR_host,
        const char* IFR_server,
        const char* interfaceMarker);
```

Description

Constructs an object from the strings 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>serverName</code>	The name of the target object's server.
<code>marker</code>	The object's marker name.
<code>IFR_host</code>	The name of a host running an Interface Repository that stores the target object's IDL definition.
<code>IFR_server</code>	The string "IFR".
<code>interfaceMarker</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 ~Object();
```

Description

Destructor.

Notes

Orbix specific.

CORBA::Object::operator=()

- Synopsis** `const Object& operator=(const Object& obj);`
- Description** Assignment operator. Copies the object `obj` to self. The reference count of self is decremented. The reference count of `obj` is incremented.
- Return Value** Returns a reference to self.
- Notes** Orbix specific.

CORBA::Object::_attachPost()

- Synopsis** `void* _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* _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

- ƒ A pointer to a filter object. The dynamic type of ƒ 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 _closeChannel(Environment& env =
 default_environment);`

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 `Status _create_request(
 Context_ptr ctx,
 const char* operation,
 NVList_ptr arg_list,
 NamedValue_ptr& result,
 Request_ptr& request,
 Flags req_flags,
 Environment& env = default_environment);`

Description Constructs a `CORBA::Request` object. Refer to `CORBA::Object::_request()` for an alternative way to create a `Request`.

Parameters

`ctx` Context object, if any, to be sent in the `Request`. If the `ctx` argument to `_create_request()` is a `nil` Context object reference, the Context may be added by calling the `ctx()` function on the `Request` object.

`operation` 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* _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*) DEREFP(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 Object_ptr _duplicate(Object_ptr obj,
 Environment& env = default_environment);`
- Description** Increments the reference count of `obj`.
- Return Value** Returns a reference to self.
- Notes** CORBA compliant.
- See Also** `CORBA::release()`

CORBA::Object::_enableInternalLock()

- Synopsis** `#ifdef _REENTRANT
Boolean _enableInternalLock(Boolean isEnabled,
 Environment& env = default_environment);
#endif`

- Description** This function is applicable only to multi-threaded versions of Orbix.

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

```
#ifdef WANT_ORBIX_FDS
int _fd(Environment& env = default_environment) const;
#endif
```

Description

Finds the UNIX 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 UNIX `select()` system 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.

This function is defined only if the following preprocessor directive is issued in the C++ file before including `CORBA.h`:

```
#define WANT_ORBIX_FDS
```

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
`ImplementationDef_ptr _get_implementation(
 Environment& env = default_environment);`

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
`InterfaceDef_ptr _get_interface(
 Environment& env = default_environment);`

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* _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* _getPre() const;`
- Description** Gets the object's pre-filter.
- Return Value** Returns a pointer to the object's pre-filter.
- Notes** Orbix specific.
- See Also** `CORBA::Object::_attachPre()`
`CORBA::Object::_getPost()`
`CORBA::Filter`

CORBA::Object::_hash()

- Synopsis** `ULong _hash(ULong maximum,
Environment& env=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

```
Boolean _hasValidOpenChannel(Environment& env =
    default_environment)
```

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* _host(
    Environment& env = default_environment) const;
```

Description Returns the name of the host on which the target object is located.

Notes Orbix specific.

CORBA::Object::_implementation()

- Synopsis** `const char* _implementation(
 Environment& env = default_environment) 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* _interfaceHost(
 Environment& env = default_environment) 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* _interfaceImplementation(
 Environment& env = default_environment) const;`
- Description** Returns the name of the Interface Repository server.
- Notes** Orbix specific.

CORBA::Object::_interfaceMarker()

- Synopsis** `const char* _interfaceMarker(
 Environment& env = default_environment) const;`
- Description** Returns the name of the target object's interface.
- Notes** Orbix specific.

CORBA::Object::_is_a()

- Synopsis** `Boolean _is_a(const char* logical_type_id,
 Environment& env = CORBA::default_environment);`
- Description** Determines if the 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** Returns 1 (true) if 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. Returns 0 (false) otherwise.
- Notes** CORBA compliant.
- See Also** `CORBA::Object::_non_existent()`

CORBA::Object::_is_equivalent()

- Synopsis** `Boolean _is_equivalent(const Object_ptr obj,
Environment& env = default_environment);`
- 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** `Boolean _isNull(
Environment& env = default_environment) 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 `true` if the object is a null object, `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**

```
Boolean _isNullProxy(
 Environment& env = default_environment) 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);
}
CATCHANY {
 . . .
}
ENDTRY
```

`pPtr` may reference a null proxy if the operation call raises an exception.

**Return Value**

Returns `true` value if the object is a null proxy and returns `false` otherwise.

**Notes**

Orbix specific.

**See Also**

`CORBA::is_nil()`

### CORBA::Object::\_isRemote()

**Synopsis**

```
virtual Boolean _isRemote(
 Environment& env = default_environment) 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 `true` if the reference is to a remote object (that is, the target of the call is a proxy), returns `false` otherwise.

**Notes**

Orbix specific.

**See Also**

`CORBA::ORB::collocated()`

**CORBA::Object::\_loader()**

- Synopsis** `virtual LoaderClass_ptr _loader(  
 Environment& env = default_environment) 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* _marker(  
 Environment& env = default_environment) const;`
- Description** Finds the target object's marker name.
- Notes** Orbix specific.
- See Also** `CORBA::Object::_marker(const char* marker)`

**CORBA::Object::\_marker()**

- Synopsis** `Boolean _marker(const char* marker,  
 Environment& env = default_environment);`
- 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 `true` if successful; returns `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 Object_ptr _nil(
 Environment& env = default_environment);
```

**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**

```
Boolean _non_existent(Environment &env =
 default_environment);
```

**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** Identifier \_object\_to\_string(  
Environment& env = default\_environment);
- Description** Converts the target object's reference to a string. An Orbix stringified object reference has the form:  
  
:\host:serverName:marker:IFR\_host:IFR\_Server  
:interfaceMarker  
  
See CORBA::ORB::object\_to\_string() for an explanation of these fields.
- Return Value** Returns a stringified object reference.
- 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** ULong \_refCount(  
Environment& env = default\_environment) 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**

```
Request_ptr _request(Identifier operation,
 Environment& env = default_environment);
```

**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 _save(
 Environment& env = default_environment);
```

**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++
class ORB : public IT_PseudoIDL {
public:
 static const ULong DEFAULT_TIMEOUT;
 static const ULong INFINITE_TIMEOUT;

 typedef string OAid;

 Object_ptr string_to_object(
 const char*,
 Environment& env = default_environment);

 Status create_list(Long count,
 NVList_ptr& new_list,
 Environment& env = default_environment) ;

 Status get_default_context(
 Context_ptr&,
 Environment& env = default_environment) ;

 Status create_operation_list(
 OperationDef_ptr,
 NVList_ptr&,
 Environment& env = default_environment) ;

 Status create_named_value(
 NamedValue_ptr&);

 Status create_environment(
 Environment_ptr& new_env,
 Environment& env = default_environment);
```

---

```
Status send_multiple_requests_oneway(
 const RequestSeq&,
 Environment& env = default_environment);

Status send_multiple_requests_deferred(
 const RequestSeq&,
 Environment& env = default_environment);

Status get_next_response(
 Request_ptr& req,
 Environment& env = default_environment);

Boolean poll_next_response(
 Environment& env = default_environment);

BOA_ptr BOA_init(int& argc,
 char** argv,
 OAid oa_identifier,
 Environment &env=IT_chooseDefaultEnv());

ObjectIdList_ptr list_initial_services(
 CORBA::Environment& env = IT_chooseDefaultEnv());

Object_ptr resolve_initial_references(
 ObjectId identifier,
 CORBA::Environment&env=IT_chooseDefaultEnv());

Object_ptr string_to_object(
 const char* host, const char* impl,
 const char* marker, const char* intfHost,
 const char* intfImpl, const char* intfMarker,
 Environment& env = default_environment);

OrbixIOCallback registerIOCallback(
 OrbixIOCallback, OrbixIOCallbackType);

ULong defaultTxTimeout(
 ULong val=INFINITE_TIMEOUT,
 Environment &env=default_environment);

Boolean collocated(
 Environment& env = default_environment) const;
Boolean collocated(
```

```
 Boolean turnOn,
 Environment& env = default_environment);
Boolean nativeExceptions(Boolean turnOn,
 Environment& env = default_environment);
Boolean nativeExceptions(Environment& env =
 default_environment) const;

const char* myHost(
 Environment& env = default_environment) const;
const char* myServer(
 Environment& env = default_environment) const;

void reSizeObjectTable(ULong,
 Environment& env = default_environment);

void setServerName(const char* serverName,
 Environment& env = default_environment);

Boolean isBaseInterfaceOf(const char* derived,
 const char* maybeABase,
 Environment& env = default_environment);
void baseInterfacesOf(IT_StringSeq&,
 const char* derived,
 Environment& env = default_environment);

Short setDiagnostics(Short level,
 Environment& env = default_environment);

Boolean pingDuringBind(Boolean,
 Environment& env = default_environment);

Boolean optimiseProtocolEncoding(Boolean,
 Environment& env = default_environment);

ULong connectionTimeout(unsigned long,
 Environment& env = default_environment);

Boolean abortSlowConnects(Boolean value);

Boolean eagerListeners(
 Boolean value,
 Environment& env = default_environment);
Boolean eagerListeners(
 Boolean value,
 Environment& env = default_environment);
```

---

```
Environment& env = default_environment);

ULong maxConnectRetries(ULong,
 Environment& env = default_environment);

Boolean noReconnectOnFailure();

Boolean resortToStatic(Boolean,
 Environment& env = default_environment);

Boolean closeChannel(int fd,
 Environment& env = IT_chooseDefaultEnv());

IT_PFV set_unsafeDelete(IT_PFV);

CORBA::Boolean bindUsingIIOP(Environment&
 env=default_environment) const;
CORBA::Boolean bindUsingIIOP(CORBA::Boolean on
 Environment& env=default_environment);
CORBA::Boolean supportBidirectionalIIOP(
 CORBA::Boolean on,
 Environment& env=default_environment);

unsigned long maxFDsPerConnectionThread();
void maxFDsPerConnectionThread(unsigned long max);
unsigned long maxConnectionThreads();
void maxConnectionThreads(unsigned long max);

CORBA::Boolean registerIOCallbackObject(
 unsigned char eventType, IT_IOCallback *obj);
CORBA::Boolean unregisterIOCallbackObject(
 unsigned char eventType, IT_IOCallback *obj);

#ifdef WANT_ORBIX_FDS
void addForeignFDSet(fd_set& theFDset, unsigned char aState);
void addForeignFD(const int fd, unsigned char aState);
void removeForeignFD(
 const int fd, unsigned char aState);
void removeForeignFDSet(
 fd_set& theFDset, unsigned char aState);
```

```
fd_set getForeignFDSet() const;
fd_set getSelectableFDSet() const;
fd_set getAllOrbixFDs() const;
CORBA::Boolean isOrbixFD(int fd);
CORBA::Boolean isForeignFD(int fd,
 unsigned char mask=FD_FOREIGN_WRITE
 | FD_FOREIGN_READ | FD_FOREIGN_EXCEPT);
CORBA::Boolean isOrbixSelectableFD(int fd);
#endif

static unsigned int GetConfigValue(
 const char* name, char*& value);

static unsigned int SetConfigValue(
 const char* name, char* value);

static void PlaceCVHandlerBefore(
 const char* before, const char* after);

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(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);

__IDL_SEQUENCE_octet * makeOrbixObjectKey(
 const char * host, const char * serverName,
 const char * interfaceName,
 Environment& env = default_environment);
```

---

```

string makeIOR(char * host, unsigned short port,
 ObjectKey objKey, char * typeID,
 Environment &env=IT_chooseDefaultEnv ());

Boolean useTransientPort(Boolean value)

void useHostNameInIOR(Boolean val,
 Environment &env=IT_chooseDefaultEnv ());

};

```

**Notes** Orbix specific.

**See Also** CORBA::BOA

## CORBA::ORB::abortSlowConnects()

**Synopsis** Boolean abortSlowConnects(  
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 `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.

Use this function selectively as it requires the use of `SIGALRM` on UNIX platforms.

**Return Value** Returns the previous setting.

**Notes** Orbix specific. It is not yet available on Multi-Thread Orbix (Orbix-MT).

**See Also** `CORBA::ORB::connectionTimeOut()`

### **CORBA::ORB::addForeignFD()**

**Synopsis**

```
void 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 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.



**Parameters**

|                     |                                                                                                                                                                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>fds</code>    | The file descriptor set to be added to the Orbix event processing loop.                                                                                                                                                                    |
| <code>aState</code> | Indicates the type of events for which the file descriptors 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::addForeignFD()`

**CORBA::ORB::ActivateCVHandler()**

**Synopsis** `static void 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 ActivateOutputHandler(  
const char* identifier);`

**Description** Activates the output handler specified in `identifier`.

**Notes** Orbix specific.

**See Also** `CORBA::UserOutput`

### **CORBA::ORB::baseInterfacesOf()**

**Synopsis**

```
void baseInterfacesOf(IT_StringSeq& interfaces,
 const char* derived,
 Environment& env = default_environment);
```

**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 bindUsingIIOP(
 Environment& env=default_environment) 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 non-zero value (true) if `_bind()` currently uses IIOP. Returns zero (false) otherwise.

**Notes**

Orbix specific.

### **CORBA::ORB::bindUsingIIOP()**

**Synopsis**

```
CORBA::Boolean bindUsingIIOP(CORBA::Boolean on,
 Environment& env=default_environment);
```

**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    A non-zero value (true) specifies that `_bind()` should use IIOp. A zero value (false) specifies that `_bind()` should use the Orbix protocol.

**Return Value** Returns the previous setting.

**Notes** Orbix specific.

## CORBA::ORB::BOA\_init()

### Synopsis

```
BOA_ptr BOA_init(int& argc,
 char** argv,
 OAid oa_identifier,
 Environment &env=IT_chooseDefaultEnv());
```

### Description

Initialises 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 archive.

`argv` A sequence of option or configuration strings that are used if `oa_identifier` is a null string. Each string is of the form:

```
-OA<suffix> <value>
```

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:

```
-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 initialises 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**

```
Boolean closeChannel(int fd,
 Environment& env = 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** Boolean collocated(  
 Environment& env = default\_environment) 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 `true` if collocation is set and returns `false` otherwise.

**Notes** Orbix specific.

**See Also** CORBA::ORB::collocated(CORBA::Boolean turnOn)  
 CORBA::Object::\_isRemote()

## CORBA::ORB::collocated()

**Synopsis** Boolean collocated(Boolean turnOn,  
 Environment& env = default\_environment);

**Description** Enforces collocation if `turnOn` is set to `true`. If collocation is enforced, binding to objects outside of the current process' address space is prevented. If set to `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** `ULong connectionTimeout(ULong timeout,  
Environment& env = default_environment);`

**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.

**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::create\_environment()**

**Synopsis** `Status create_environment(  
Environment_ptr& new_env,  
Environment& env = default_environment);`

**Description** Returns a newly-created `Environment` in the parameter `new_env`.

**Return Value** Returns true (1) if successful, false (0) otherwise.

---

**Notes** CORBA compliant.

**See Also** CORBA::Environment

### CORBA::ORB::create\_list()

**Synopsis** Status create\_list(Long count,  
NVList\_ptr& new\_list,  
Environment& env = default\_environment);

**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\_lis A pointer to the start of the list. The caller must release the  
t reference when it is no longer needed, or assign it to a NVList\_var variable for automatic management.

**Return Value** Returns true (1) if successful, false (0) 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** Status create\_named\_value(NamedValue\_ptr& rNamedValue );

**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 true (1) if successful, false (0) 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

```
Status create_operation_list(
 OperationDef_ptr operation,
 NVList_ptr list);
```

#### Description

Returns an `NVList`, in the parameter `list`, initialised 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

`operation` A reference to the Interface Repository object describing the operation.

`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 true (1) if successful, false (0) otherwise.

**Notes** CORBA compliant. Use of this function requires a program to be linked with the `IRclt` library and for the relevant IDL file to be compiled with the `-R` switch.

**See Also** `CORBA::NVList`  
`CORBA::Any`



---

```
CORBA::create_list()
CORBA::NamedValue
```

## CORBA::ORB::DeactivateCVHandler()

- Synopsis** `static void 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 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 ULong 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::defaultTxTimeout()**

#### **Synopsis**

```
ULong defaultTxTimeout(
 ULong val = INFINITE_TIMEOUT,
 Environment& env = default_environment);
```

#### **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.

**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::Environment::timeout()`

### **CORBA::ORB::eagerListeners()**

#### **Synopsis**

```
Boolean eagerListeners(
 Boolean value,
 Environment& env = default_environment);
```

#### **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 behaviour.

This feature is not necessary in multi-threaded versions of Orbix.

### Parameters

**value** 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 behaviour.

**Return Value** Returns the previous value.

**Notes** Orbix specific.

**See Also** `Boolean eagerListeners( Environment& env = default_environment )`  
`CORBA::ORB::connectionTimeout()`

## CORBA::ORB::eagerListeners()

**Synopsis** `Boolean eagerListeners( Environment& env = default_environment );`

**Description** Determines if eager listening is set. Refer to the overloaded modifier function for details.

**Notes** Orbix specific.

**See Also** `Boolean eagerListeners( Boolean value, Environment& env = default_environment )`

## CORBA::ORB::getAllOrbixFDs()

**Synopsis** `fd_set getAllOrbixFDs() const;`

**Description** Returns the current set of all Orbix file descriptors. In Orbix-MT, 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::get\_default\_context()**

**Synopsis**

```
Status get_default_context(
 Context_ptr& context,
 Environment& env = default_environment);
```

**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::getForeignFDSet()**

**Synopsis**

```
fd_set 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::get\_next\_response()

- Synopsis** `Status get_next_response(  
Request_ptr& req,  
Environment& env = default_environment);`
- 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::getSelectableFDSet()

- Synopsis** `fd_set getSelectableFDSet() const;`
- Description** Returns the set of Orbix file descriptors that are guaranteed to be active when Orbix has events to process. In Orbix-MT, 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::GetConfigValue()**

**Synopsis**

```
static unsigned int GetConfigValue(
 const char* name, char*& value);
```

**Description**

Obtains the value of the configuration entry named in name.

**Parameters**

name    The name of the desired configuration entry, for example,  
         IT\_DAEMON\_PORT.

value   The value of the configuration entry.

**Return Value**

Returns 1 (true) if a value for the configuration entry specified in name is available, returns 0 (false) otherwise.

**Notes**

Orbix specific.

**See Also**

CORBA::ORB::SetConfigValue()  
CORBA::ORB::ReinitialiseConfig()  
CORBA::UserCVHandler  
CORBA::ExtraRegistryCVHandler  
CORBA::ExtraConfigFileHandler

### **CORBA::ORB::INFINITE\_TIMEOUT**

**Synopsis**

```
static const ULong 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** `Boolean isBaseInterfaceOf(const char* derived,  
const char* maybeABase,  
Environment& env = default_environment);`
- Description** Determines whether the interface named in `maybeABase` is a base interface of the interface named in `derived`.
- Return Value** Returns true if `maybeABase` is a base interface of `derived` (or `maybeABase` and `derived` are the same interface) and returns false otherwise.
- Notes** Orbix specific.
- See Also** `CORBA::ORB::baseInterfacesOf()`

## CORBA::ORB::isForeignFD()

- Synopsis** `CORBA::Boolean isForeignFD(int fd,  
unsigned char mask=FD_FOREIGN_WRITE  
| FD_FOREIGN_READ | FD_FOREIGN_EXCEPT);`
- Description** Tests if a specified file descriptor has been registered as a foreign file descriptor with the Orbix event processing loop.
- Parameters**
- `fd` The file descriptor to be tested for membership of the registered file descriptor set.
  - `mask` This parameter allows you to test for which type of events the file descriptor is monitored. The value can be `FD_FOREIGN_WRITE`, `FD_FOREIGN_READ`, `FD_FOREIGN_EXCEPT`, or a combination of these.
- Return Value** Returns a non-zero value (true) if the specified file descriptor is part of the foreign file descriptor set *and* is monitored for the specified events. Returns zero (false) otherwise.
- Notes** Orbix specific.

### **CORBA::ORB::isOrbixFD()**

- Synopsis** `CORBA::Boolean 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 a non-zero value (true) if the specified file descriptor is part of the Orbix file descriptor set. Returns zero (false) otherwise.
- Notes** Orbix specific.

### **CORBA::ORB::isOrbixSelectableFD()**

- Synopsis** `CORBA::Boolean 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 a non-zero value (true) if the specified file descriptor is part of the Orbix selectable file descriptor set. Returns zero (false) otherwise.
- Notes** Orbix specific.



## CORBA::ORB::list\_initial\_services()

- Synopsis** `ObjectIdList_ptr list_initial_services(  
CORBA::Environment& env = 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 makeIOR(char * host, unsigned short port,  
ObjectKey objKey, char * typeID,  
Environment &env=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**

```
_IDL_SEQUENCE_octet * makeOrbixObjectKey(
 const char * host, const char * serverName,
 const char * interfaceName,
 Environment& env = default_environment);
```

**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::maxConnectionThreads()**

**Synopsis**

```
unsigned long maxConnectionThreads();
```

**Description**

As described in the *Orbix C++ Programmer's Guide*, Orbix-MT creates internal connection threads to process incoming requests to servers. This function returns the maximum number of internal connection threads created by Orbix-MT in the current process.

**Return Value**

Returns the current maximum number of connection threads.

**Notes**

Orbix specific.

**See Also**

`CORBA::maxFDsPerConnectionThread()`

## CORBA::ORB::maxConnectionThreads()

- Synopsis** `void maxConnectionThreads(unsigned long max);`
- Description** As described in the *Orbix C++ Programmer's Guide*, Orbix-MT creates internal connection threads to process incoming requests to servers. This function sets the maximum number of internal connection threads created by Orbix-MT in the current process.
- Parameters**
- `max` The maximum number of connection threads to be created.
- Notes** Orbix specific.
- See Also** `CORBA::ORB::maxFDsPerConnectionThread()`  
`CORBA::ORB::removeForeignFD()`  
`CORBA::ORB::removeForeignFDSet()`

## CORBA::ORB::maxConnectRetries()

- Synopsis** `ULong maxConnectRetries(ULong retries,  
Environment& env = default_environment);`
- Description** If an operation call cannot be made on the first attempt because the transport (for example, 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 `IT_CONNECT_ATTEMPTS` entry in the Orbix configuration file 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 `IT_CONNECT_ATTEMPTS` value is used only if `maxConnectRetries()` is set to zero.
- 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::maxFDsPerConnectionThread()**

**Synopsis** `unsigned long maxFDsPerConnectionThread();`

**Description** As described in the *Orbix C++ Programmer's Guide*, Orbix-MT creates internal connection threads to process incoming requests to servers. This function returns the current maximum number of connections per connection thread.

**Return Value** Returns the current maximum value.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::maxConnectionThreads()`

### **CORBA::ORB::maxFDsPerConnectionThread()**

**Synopsis** `void maxFDsPerConnectionThread(unsigned long max);`

**Description** As described in the *Orbix C++ Programmer's Guide*, Orbix-MT creates internal connection threads to process incoming requests to servers. This function allows you to specify the maximum number of connections to be serviced by each connection thread. When a connection thread reaches the maximum value, Orbix-MT creates a new connection thread.

#### **Parameters**

`max` The maximum number of connections to be serviced by each internal connection thread.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::maxConnectionThreads()`

### **CORBA::ORB::myHost()**

**Synopsis** `const char* myHost(  
 Environment& env = default_environment) 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* myServer(
 Environment& env = default_environment) 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::nativeExceptions()

**Synopsis**

```
Boolean nativeExceptions(Environment& env =
 default_environment) const;
```

**Description** Returns 1 (true) if Orbix throws a C++ exception to the caller of a remote operation and returns 0 (false) otherwise.

**Notes** Orbix specific.

**See Also** CORBA::ORB::nativeExceptions(Boolean turnOn,  
Environment&)

### CORBA::ORB::nativeExceptions()

**Synopsis**

```
Boolean nativeExceptions(Boolean turnOn,
 Environment& env = default_environment);
```

**Description** Requests Orbix not to throw a C++ exception to the caller of a remote operation if turnOn is 0 (false). By default Orbix throws C++ exceptions when the host compiler supports exception handling.

**Notes** Orbix specific.

**See Also** CORBA::ORB::nativeExceptions(Environment&) const

### **CORBA::ORB::noReconnectOnFailure()**

#### **Synopsis**

```
Boolean noReconnectOnFailure(Boolean value,
 Environment &env=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.

With single-threaded Orbix, 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:** Multi-threaded Orbix (OrbixMT) automatically tries to re-establish without throwing a system exception.

---

You can change this default behaviour 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**

`value` A value of 1 (true) means that all client attempts to contact a server subsequent to closure of the communications channel raise a `CORBA::COMM_FAILURE` system exception.

A value of 0 (false) means that Orbix attempts to re-establish a failed connection to a server on the next invocation by a client. This is the default behaviour.

**Return Value** Returns the previous value.

**Notes** Orbix specific.

**See Also** `CORBA::ORB::maxConnectRetries()`

## CORBA::ORB::object\_to\_string()

**Synopsis** `char* object_to_string(Object_ptr obj,  
Environment& env = default_environment);`

**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:

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>host</code>             | 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**

```
Boolean optimiseProtocolEncoding(
 Boolean value,
 Environment& env = default_environment);
```

**Description**

Enables protocol encoding optimisations if `value` is set to `true`, which is the default. 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::Output()

**Synopsis**

```
static void Output(char* message, int level = 1);
static void Output(Environment& env, int level = 1);
static void Output(SystemException* sysEx, 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**

`message` Outputs a string.  
`env` Outputs details of an Environment.  
`sysEx` Outputs details of a system exception.  
`level` The diagnostic level. (All Orbix output is assigned level 1.)

**Notes**

Orbix specific.

**See Also**

`CORBA::UserOutput`  
`CORBA::ORB::setDiagnostics`



---

## CORBA::ORB::pingDuringBind()

**Synopsis**

```
Boolean pingDuringBind(Boolean pingOn,
 Environment& env = default_environment);
```

**Description**

By default, `_bind()` raises an exception if the object on which the `_bind()` 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 recognises the target object. You can disable pinging using `pingDuringBind()`, by passing `false` to the parameter `pingOn`.

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).
- ◆ If a host name is specified to `_bind()`, the `_bind()` does not itself make any remote calls; it simply sets up a proxy with the required fields.
- ◆ If a host name is not specified, Orbix uses its locator to find a suitable server, but `_bind()` does not interact with that server to determine if the required object exists within it.

**Return Value**

Returns the previous setting.

**Notes**

Orbix specific.

## CORBA::ORB::PlaceCVHandlerAfter()

**Synopsis**

```
static void 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

Orbis specific.

### See Also

`CORBA::ORB::PlaceCVHandlerBefore()`  
`CORBA::ORB::ReinitialiseConfig()`  
`CORBA::UserCVHandler`  
`CORBA::ExtraRegistryCVHandler`  
`CORBA::ExtraConfigFileHandler`

## **CORBA::ORB::PlaceCVHandlerBefore()**

### Synopsis

```
static void 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

Orbis specific.

---

**See Also** CORBA::ORB::PlaceCVHandlerAfter()  
CORBA::ORB::ReinitialiseConfig()  
CORBA::UserCVHandler  
CORBA::ExtraRegistryCVHandler  
CORBA::ExtraConfigFileHandler

## CORBA::ORB::poll\_next\_response()

**Synopsis** Boolean poll\_next\_response(  
Environment& env = default\_environment);

**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** OrbixIOCallback registerIOCallback(  
OrbixIOCallback IOCallback,  
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++
env.m_request->descriptor();
```

assuming that `env` is the formal `CORBA::Environment` parameter of the current operation. 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 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**

`eventType`      Orbix can monitor both native Orbix file descriptors and foreign file descriptors for events, as described in `CORBA::ORB::addForeignFd()`. This parameter indicates which file descriptors should be monitored with respect to the callback object being registered. The value of this parameter is `CORBA::OrbixIO`, `CORBA::ForeignIO`, or a logical combination of these.

`obj`             The callback object to be registered with Orbix.

**Return Value**   Returns a non-zero (true) value if the object is successfully registered. Returns zero (false) otherwise.

**Notes**            Orbix specific.

**See Also**        `CORBA::ORB::addForeignFD()`  
                 `CORBA::ORB::unregisterIOCallback()`  
                 `CORBA::IT_IOCallback`

### **CORBA::ORB::removeForeignFD()**

**Synopsis**

```
void 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**

|                     |                                                                                                                                                                                                                                  |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>fd</code>     | The file descriptor to be removed from the Orbix event processing loop.                                                                                                                                                          |
| <code>aState</code> | Indicates the type of events for which the file descriptor should no longer be monitored by Orbix. This can be <code>FD_READ</code> , <code>FD_WRITE</code> , <code>FD_EXCEPT</code> , 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 removeForeignFDSet(
 fd_set& fds, 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**

|                     |                                                                                                                                                                                                                                   |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>fds</code>    | The file descriptor set to be removed from the Orbix event processing loop.                                                                                                                                                       |
| <code>aState</code> | Indicates the type of events for which the file descriptors should no longer be monitored by Orbix. This can be <code>FD_READ</code> , <code>FD_WRITE</code> , <code>FD_EXCEPT</code> , or a logical combination of these values. |

**Notes**

Orbix specific.

**See Also** CORBA::ORB::addForeignFD()  
 CORBA::ORB::addForeignFDSet()  
 CORBA::ORB::removeForeignFD()

## CORBA::ORB::ReinitialiseConfig()

**Synopsis** static void 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::reSizeObjectTable()

**Synopsis** void reSizeObjectTable(ULong newSize,  
 Environment& env = default\_environment);

**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

```
Object_ptr resolve_initial_references(
 ObjectId identifier,
 CORBA::Environment&env=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**

```
Boolean resortToStatic(Boolean value,
 Environment& env = default_environment);
```

**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 `false` for the first parameter. The default setting is `true`. Setting the value to `false` means that Orbix raises an exception if the server or client is not linked with the actual (dynamic) proxy code.

**Return Value** Returns the previous setting.

**Notes** Orbix specific.

### **CORBA::ORB:: send\_multiple\_requests\_deferred()**

**Synopsis**      `Status send_multiple_requests_deferred(  
                  const RequestSeq& requests,  
                  Environment& env = default_environment);`

**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**      Returns 1 (true) if successful, returns 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**      `Status send_multiple_requests_oneway(  
                  const RequestSeq& requests,  
                  Environment& env = default_environment);`

**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** Returns true (1) if successful, false (0) otherwise.

**Notes** CORBA compliant.

**See Also** `CORBA::Request::send_oneway()`  
`CORBA::ORB::send_multiple_requests_deferred()`

**CORBA::ORB::set\_unsafeDelete()**

**Synopsis** `IT_PFV 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::SetConfigValue()

**Synopsis**     static unsigned int SetConfigValue(  
                  const char\* name, char\* value);

**Description**   Sets the value of the configuration entry named in *name* for this process. (It does not set the configuration entry in the Orbix configuration file or system registry.)

The configuration entries that can be set by `SetConfigValue()` are:

- ◆ IT\_DAEMON\_SERVER\_BASE
- ◆ IT\_DAEMON\_SERVER\_RANGE
- ◆ IT\_DAEMON\_PORT
- ◆ IT\_ERRORS
- ◆ IT\_IMP\_REP\_PATH
- ◆ IT\_LOCATOR\_PATH
- ◆ IT\_INT\_REP\_PATH
- ◆ IT\_LOCAL\_DOMAIN

The function `ReinitialiseConfig()` invalidates the effect of a call to this function.

### Parameters

*name*   The name of the configuration entry, for example, `IT_INT_REP_PATH`.  
*value*   The value to be assigned to the configuration entry.

**Return Value**   Returns 1 (true) if successful; returns 0 (false) otherwise.

**Notes**           Orbix specific.

**See Also**       CORBA::ORB::GetConfigValue()  
                  CORBA::ORB::ReinitialiseConfig()  
                  CORBA::UserCVHandler  
                  CORBA::ExtraRegistryCVHandler  
                  CORBA::ExtraConfigFileHandler

---

## CORBA::ORB::setDiagnostics()

- Synopsis** `Short setDiagnostics(Short level,  
Environment& env = default_environment);`
- Description** Controls the level of diagnostic messages output to the `cout` stream by the Orbix libraries. The previous setting is returned.
- | Level | Output                                   |
|-------|------------------------------------------|
| 0     | No diagnostics                           |
| 1     | Simple diagnostics (this is the default) |
| 2     | Full diagnostics                         |
- 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::setServerName()

- Synopsis** `void setServerName(const char* serverName,  
Environment& env = default_environment);`
- 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::string\_to\_object()**

**Synopsis**

```
Object_ptr string_to_object(const char* obj_ref_string,
 Environment& env = default_environment);
```

**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 recognised stringified object reference format.

**Notes**

CORBA compliant.

**See Also**

```
CORBA::ORB::string_to_object(
 const char* host,
 const char* serverName,
 const char* marker,
 const char* IFR_host,
 const char* IFR_server,
 const char* interfaceMarker),
CORBA::ORB::object_to_string()
CORBA::ORB::pingDuringBind()
CORBA::Object::Object(const char* obj_ref_string);
```

### **CORBA::ORB::supportBidirectionalIIOP()**

**Synopsis**

```
CORBA::Boolean supportBidirectionalIIOP(
 CORBA::Boolean on,
 Environment& env=default_environment);
```

**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 IONA Technologies' 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    A non-zero value (true) enables bidirectional IIOp. A zero value (false) disables bidirectional IIOp. By default, bidirectional IIOp is disabled.

**Return Value** Returns the previous setting.

**Notes** Orbix specific.

## CORBA::ORB::string\_to\_object()

### Synopsis

```
Object_ptr string_to_object(
 const char* host,
 const char* serverName,
 const char* marker,
 const char* IFR_host,
 const char* IFR_server,
 const char* interfaceMarker,
 Environment& env = default_environment);
```

**Description** Creates an object from the strings given as arguments to an object reference.

### Parameters

|            |                                                                                                    |
|------------|----------------------------------------------------------------------------------------------------|
| host       | The host name of the target object.                                                                |
| serverName | The name of the target object's server.                                                            |
| marker     | The object's marker name.                                                                          |
| IFR_host   | The name of a host running an Interface Repository that stores the target object's IDL definition. |
| IFR_server | The string "IFR".                                                                                  |

`interfaceMarker`      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(const char* host,`  
    `const char* serverName,`  
    `const char* marker,`  
    `const char* IFR_host,`  
    `const char* IFR_server,`  
    `const char* interfaceMarker)`

### **CORBA::ORB::unregisterIOCallbackObject()**

**Synopsis**            `CORBA::Boolean unregisterIOCallbackObject(`  
    `unsigned char eventType, IT_IOCallback *obj);`

**Description**        This function removes a callback object registered using `CORBA::ORB::registerIOCallbackObject()`.

#### **Parameters**

`eventType`            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 `CORBA::OrbixIO`, `CORBA::ForeignIO`, or a logical combination of these.

`obj`                    The callback object to be unregistered.

**Return Value**        Returns a non-zero (true) value if the object is successfully unregistered. Returns zero (false) otherwise.

**Notes**              Orbix specific.

**See Also**            `CORBA::ORB::addForeignFD()`  
`CORBA::ORB::registerIOCallback()`  
`CORBA::IT_IOCallback`



---

## CORBA::ORB::useTransientPort()

**Synopsis** `Boolean useTransientPort(Boolean value);`

**Description** An IOR by default contains the daemon's port; however this behaviour is configurable. You can replace the daemon's port number with the server's transient port by calling this API. In addition, you can use the `-port` switch with the `putit` utility to specify the actual port that you want the server to run on. If this switch is not used, the daemon assigns the port.

Calling `CORBA::Orbix.useTransientPort(1)` sets the IOP port to the port that the server is actually listening on.

**Return Value** Returns the previous setting.

**See Also** `CORBA::ORB::makeIOR()`  
`CORBA::ORB::useHostNameInIOR()`

## CORBA::ORB::useHostNameInIOR()

**Synopsis** `void useHostNameInIOR(Boolean val,  
Environment &env=IT_chooseDefaultEnv ());`

**Description** By default a server exports IORs containing the host name as given by the system call `gethostname()`. This API configures the server to put the hostname or IP address into the IOR. To set export the host name in the IOR, set the parameter `val` to `TRUE`; to export IP address, set it to `FALSE`.

**See Also** `CORBA::ORB::makeIOR()`  
`CORBA::ORB::useTransientPort()`



# 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,  
 Environment& env = default_environment);  
  
 static Principal_ptr _duplicate(  
 Principal_ptr obj,  
 Environment& env = default_environment);  
  
 static Principal_ptr _nil(  
 Environment& env = default_environment);  
};`

**Notes** CORBA compliant.

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

## CORBA::Principal::Principal()

**Synopsis** `Principal();`

**Description** Default constructor.

**Notes** Orbix specific.

**See Also** `CORBA::Principal::IT_create()`

### **CORBA::Principal::\_duplicate()**

- Synopsis**      `static Principal_ptr _duplicate(  
                  Principal_ptr obj,  
                  Environment& env = default_environment);`
- 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 Principal_ptr _nil(  
                  Environment& env = default_environment);`
- Description**    Returns a nil object reference for a `Principal` object.
- Notes**          CORBA compliant.
- See Also**        `CORBA::is_nil()`

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

- Synopsis**      `static Principal_ptr IT_create(  
                  const char* obj,  
                  Environment& env = default_environment);`
- 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 Request : public IT_PseudoIDL {
public:
 Status invoke();

 Status get_response();

 Object_ptr target(
 Environment& env = default_environment) const;
```

```
const char* operation(
 Environment& env = default_environment) const;

NVList_ptr arguments(
 Environment& env = default_environment);

NamedValue_ptr result(
 Environment& env = default_environment);

Environment_ptr env(
 Environment& env = default_environment);

void ctx(Context_ptr IT_cp,
 Environment& env = default_environment);

Context_ptr ctx(
 Environment& env = default_environment) const;

Status send_oneway(
 Environment& env = default_environment);

Status send_deferred(
 Environment& env = default_environment);

Boolean poll_response(
 Environment& env = default_environment);

Request();

Request(Object_ptr target,
 const Identifier OperationName = 0,
 Environment& env = default_environment);

virtual ~Request();

void reset(ObjectRef obj=0,
 const char* OperationName = 0);
void reset(const char* OperationName);
void setOperation (const char *opname);

void setTarget(ObjectRef target);
```

---

```
#ifdef WANT_ORBIX_FDS
 int descriptor(void) const;
#endif

Request& operator<<(const int& i);
Request& operator<<(const Boolean& o);
Request& operator<<(const Short& s);
Request& operator<<(const Long& l);
Request& operator<<(const LongLong& ll);
Request& operator<<(const UShort& us);
Request& operator<<(const ULong& ul);
Request& operator<<(const ULongLong& ull);
Request& operator<<(const Float& f);
Request& operator<<(const Double& d);
Request& operator<<(const 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<<(const IT_FixedBase& f);

Request& insertOctet(const Octet&);

Request& operator>>(Short& s);
Request& operator>>(Long& l);
Request& operator>>(LongLong& ll);
Request& operator>>(UShort& us);
Request& operator>>(ULong& ul);
Request& operator>>(ULongLong& ull);
Request& operator>>(Float& f);
Request& operator>>(Double& d);
Request& operator>>(Char& c);
Request& operator>>(Boolean& o);
Request& operator>>(char*& s);
Request& operator>>(ObjectRef&);
Request& operator>>(any&);
Request& operator>>(IT_FixedBase& f);

Request& extractOctet(Octet&);

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 encodeFixedArray(IT_FixedBase*& s, ULong len);
void decodeFixedArray(IT_FixedBase*& 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);
```

```
static Request_ptr IT_create(
 Environment& env = default_environment);

static Request_ptr IT_create(Object_ptr target,
 const Identifier OperationName=0,
 Environment& env = default_environment,
 Boolean x=0, Boolean y=0);

static Request_ptr _duplicate(
 Request_ptr obj,
 Environment& env = default_environment);

static Request_ptr _nil(
 Environment& env = default_environment);
};
```

### Notes

CORBA compliant.



## CORBA::Request::Request()

- Synopsis** `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** `Request(Object_ptr target,  
const Identifier OperationName = 0,  
Environment& env = default_environment);`
- 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()**

|                    |                                  |
|--------------------|----------------------------------|
| <b>Synopsis</b>    | <code>virtual ~Request();</code> |
| <b>Description</b> | Destructor.                      |
| <b>Notes</b>       | Orbix specific.                  |

### **CORBA::Request::operator>>()**

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b> | <pre>Request&amp; operator&gt;&gt;(Short&amp;); Request&amp; operator&gt;&gt;(Long&amp;); Request&amp; operator&gt;&gt;(LongLong&amp;); Request&amp; operator&gt;&gt;(UShort&amp;); Request&amp; operator&gt;&gt;(ULong&amp;); Request&amp; operator&gt;&gt;(ULongLong&amp;); Request&amp; operator&gt;&gt;(Float&amp;); Request&amp; operator&gt;&gt;(Double&amp;); Request&amp; operator&gt;&gt;(Char&amp;); Request&amp; operator&gt;&gt;(Boolean&amp;); Request&amp; operator&gt;&gt;(char*&amp;); Request&amp; operator&gt;&gt;(Object_ptr&amp;); Request&amp; operator&gt;&gt;(Any&amp;); Request&amp; operator&gt;&gt;(IT_FixedBase&amp;);</pre> |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                    |                                                                                                                                                                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Once an invocation has been made, you can examine the operation's return value using the extraction operator, <code>operator&gt;&gt;()</code> . 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()`.

|                     |                              |
|---------------------|------------------------------|
| <b>Return Value</b> | Returns a reference to self. |
|---------------------|------------------------------|

|              |                                                                                         |
|--------------|-----------------------------------------------------------------------------------------|
| <b>Notes</b> | Orbix specific. The CORBA compliant function is <code>CORBA::Request::result()</code> . |
|--------------|-----------------------------------------------------------------------------------------|

|                 |                                                                                                                   |
|-----------------|-------------------------------------------------------------------------------------------------------------------|
| <b>See Also</b> | <pre>CORBA::Request::result() CORBA::Request::extractOctet() CORBA::Request::decodeArray() CORBA::extract()</pre> |
|-----------------|-------------------------------------------------------------------------------------------------------------------|

## CORBA::Request::operator<<()

### Synopsis

```
Request& operator<<(const Boolean&);
Request& operator<<(const Short&);
Request& operator<<(const Long&);
Request& operator<<(const LongLong&);
Request& operator<<(const UShort&);
Request& operator<<(const ULong&);
Request& operator<<(const ULongLong&);
Request& operator<<(const Float&);
Request& operator<<(const Double&);
Request& operator<<(const Char&);
Request& operator<<(const char*&);
Request& operator<<(const Context&);
Request& operator<<(const Flags&);
Request& operator<<(Object_ptr const&);
Request& operator<<(Any&);
Request& operator<<(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 << 1
 << 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 Request_ptr _duplicate(
 Request_ptr obj,
 Environment& env = default_environment);
```

**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 Request_ptr _nil(
 Environment& env = default_environment);
```

**Description** Returns a nil object reference for a Request.

**Notes** Orbix specific.

**See Also** CORBA::is\_nil()

### CORBA::Request::arguments()

**Synopsis**

```
NVList_ptr arguments(
 Environment& env = default_environment);
```

**Description** Returns the arguments to the Request's operation in an NVList.

**Notes** CORBA compliant.

**See Also** CORBA::NVList

### CORBA::Request::assumeArgsOwnership()

**Synopsis**

```
void assumeArgsOwnership(CORBA::Boolean val);
```

**Description** Specifies whether a CORBA::Request object should assume responsibility for the memory associated with the `arg_list` parameter passed to `CORBA::Object::_create_request()`. By default, a CORBA::Request object does not assume ownership of this memory.

#### Parameters

`val` A non-zero (true) value indicates that the CORBA::Request object should assume ownership of the `arg_list` parameter. A zero (false) value indicates that you should manage the memory for this parameter.

**Notes** CORBA compliant.

**See Also** `CORBA::Object::_create_request()`

### **CORBA::Request::assumeResultOwnership()**

**Synopsis** `void assumeResultOwnership(CORBA::Boolean val);`

**Description** Specifies whether a `CORBA::Request` object should assume responsibility for the memory associated with the `result` parameter passed to `CORBA::Object::_create_request()`. By default, a `CORBA::Request` object does not assume ownership of this memory.

#### **Parameters**

`val` A non-zero (true) value indicates that the `CORBA::Request` object should assume ownership of the `result` parameter. A zero (false) value indicates that the programmer should manage the memory for this parameter.

**Notes** CORBA compliant.

**See Also** `CORBA::Object::_create_request()`

### **CORBA::Request::ctx()**

**Synopsis** `Context_ptr ctx(  
    Environment& env = default_environment) 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 ctx(Context_ptr c,  
    Environment& env = default_environment);`

|                    |                                                              |
|--------------------|--------------------------------------------------------------|
| <b>Description</b> | Inserts a Context into a request.                            |
| <b>Notes</b>       | CORBA compliant.                                             |
| <b>See Also</b>    | CORBA::Request::ctx(CORBA::Context_ptr c);<br>CORBA::Context |

## CORBA::Request::decodeArray()

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b> | <pre>void decodeCharArray(char*&amp;, ULong&amp; len); void decodeUCharArray(unsigned char*&amp;, ULong&amp; len); void decodeShortArray(Short*&amp;, ULong&amp; len); void decodeUShortArray(UShort*&amp;, ULong&amp; len); void decodeLongArray(Long*&amp;, ULong&amp; len); void decodeULongArray(ULong*&amp;, ULong&amp; len); void decodeLongLongArray(LongLong*&amp;, ULong&amp; len); void decodeULongLongArray(ULongLong*&amp;, ULong&amp; len); void decodeFloatArray(Float*&amp;, ULong&amp; len); void decodeDoubleArray(Double*&amp;, ULong&amp; len); void decodeOctetArray(Octet*&amp;, ULong&amp; len); void decodeBooleanArray(Boolean*&amp;, ULong&amp; len); void decodeFixedArray(IT_FixedBase*&amp;, ULong&amp; len);</pre> |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                    |                                                                                                                                                                                                                                                                                                                                       |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | 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. |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                   |                                                               |
|-------------------|---------------------------------------------------------------|
| <b>Parameters</b> | <p>len Contains the length of the array (after the call).</p> |
|-------------------|---------------------------------------------------------------|

|              |                                                                           |
|--------------|---------------------------------------------------------------------------|
| <b>Notes</b> | Orbix specific. The CORBA compliant function is CORBA::Request::result(). |
|--------------|---------------------------------------------------------------------------|

|                 |                                                                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>See Also</b> | CORBA::Request::result()<br>CORBA::Request::encodeArray()<br>CORBA::Request::operator>>()<br>CORBA::Request::extractOctet()<br>CORBA::extract() |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------|

### CORBA::Request::descriptor()

**Synopsis**

```
#ifdef WANT_ORBIX_FDS
 int descriptor(void) const;
#endif
```

**Description**

Returns the file descriptor associated with a request. This function is defined only if the following C++ preprocessor directive is issued in the C++ file before including CORBA.h.

```
#define WANT_ORBIX_FDS
```

**Notes**

Orbix specific.

### CORBA::Request::encodeArray()

**Synopsis**

```
void encodeCharArray(char*&, ULong len);
void encodeUCharArray(unsigned char*&, ULong len);
void encodeShortArray(Short*&, ULong len);
void encodeUShortArray(UShort*&, ULong len);
void encodeLongArray(Long*&, ULong len);
void encodeULongArray(ULong*&, ULong len);
void encodeLongLongArray(LongLong*&, ULong& len);
void encodeULongLongArray(ULongLong*&, ULong& len);
void encodeFloatArray(Float*&, ULong len);
void encodeDoubleArray(Double*&, ULong len);
void encodeOctetArray(Octet*&, ULong len);
void encodeBooleanArray(Boolean*&, ULong len);
void encodeFixedArray(IT_FixedBase*&, 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::env()

- Synopsis** `Environment_ptr env(  
 Environment& env = default_environment);`
- 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** `Request& extractOctet(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** `Status 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** `Request& insertOctet(const 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** `Status 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 Request_ptr IT_create(Object_ptr target,
 const Identifier OperationName = 0,
 Environment& env = default_environment);
static Request_ptr IT_create(
 Environment& env = default_environment);
```

**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* operation(
 Environment& env = default_environment) const;
```

**Description**

Gets the `Request`'s operation name.

**Notes**

CORBA compliant.

**See Also**

`CORBA::Request::setOperation()`

### **CORBA::Request::poll\_response()**

**Synopsis**

```
Boolean poll_response(
 Environment& env = default_environment);
```

**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 reset(const char* operationName);
void reset(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**

|                            |                                                                   |
|----------------------------|-------------------------------------------------------------------|
| <code>obj</code>           | The target object. (If set to 0, the target object is not reset.) |
| <code>operationName</code> | The operation name.                                               |

**Notes**

Orbix specific.

**CORBA::Request::result()**

- Synopsis** `NamedValue_ptr result(  
 Environment& env = default_environment);`
- 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** `Status send_deferred(  
 Environment& env = default_environment);`
- 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**

```
Status send_oneway(
 Environment& env = default_environment);
```

**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 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 set_return_type(TypeCode_ptr tc,  
Environment& env = default_environment);`

**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 setTarget(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(  
Environment& env = default_environment) 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(
 Environment& IT_env = default_environment) const = 0;

 virtual Context_ptr ctx(
 Environment& IT_env = default_environment) const = 0;

 virtual void params(NVList_ptr,
 Environment& IT_env = default_environment) = 0;

 virtual Any* result(
 Environment& IT_env = default_environment) = 0;

 virtual OperationDef_ptr op_def(
 Environment& IT_env = default_environment) = 0;

 virtual void result(CORBA::Any*,
 Environment& IT_env = default_environment) = 0;

 virtual void exception(CORBA::Any*,
 Environment& IT_env = default_environment) = 0;

 virtual Object_ptr target(
 Environment& IT_env = default_environment) const = 0;

 virtual const char* operation(
 Environment& IT_env = default_environment) const = 0;

 virtual NVList_ptr arguments(
 Environment& IT_env = default_environment) = 0;

 virtual void arguments (NVList_ptr,
 Environment& IT_env = default_environment) = 0;

 virtual Environment_ptr env(
 Environment& IT_env = default_environment) = 0;
 virtual void env(Environment_ptr,
 Environment& IT_env = default_environment) = 0;
```

```
protected:
 ServerRequest();
 virtual ~ServerRequest();
 ServerRequest* operator&();
 const ServerRequest* operator&() const;
};
```

**Notes** CORBA compliant.

**See Also** CORBA::DynamicImplementation

### **CORBA::ServerRequest::ServerRequest()**

**Synopsis** 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 ~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** arguments(NVList\_ptr nvl,  
Environment& IT\_env = default\_environment) = 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**

```
Context_ptr ctx(
 Environment& IT_env = default_environment) 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 exception(CORBA::Any*,
 Environment& IT_env = default_environment) = 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** Environment\_ptr env(  
Environment& IT\_env = default\_environment);
- 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 env(Environment\_ptr,  
Environment& IT\_env = default\_environment) = 0;
- Description** Sets the Environment associated with the ServerRequest.
- Notes** Orbix specific.
- See Also** CORBA::Environment()

**CORBA::ServerRequest::op\_def()**

- Synopsis** OperationDef\_ptr op\_def(  
Environment& IT\_env = default\_environment) 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* op_name(
 Environment& IT_env = default_environment) 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* operation(
 Environment& IT_env = default_environment) 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 params(NVList_ptr nvl,  
Environment& IT_env = default_environment) 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** `Any* result(  
Environment& IT_env = default_environment) = 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**

```
Object_ptr target(
 Environment& IT_env = default_environment) 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::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** `String_var();`  
**Description** Default constructor.  
**Notes** CORBA compliant.  
**See Also** Other constructors.

### **CORBA::String\_var::String\_var()**

**Synopsis** `String_var(char* p);`  
**Description** Conversion from a `char*`. The `String_var` assumes ownership of the parameter `p`.  
**Notes** CORBA compliant.  
**See Also** Other `String_var` constructors.

### **CORBA::String\_var::String\_var()**

**Synopsis** `String_var(const String_var& s);`  
**Description** Copy constructor.  
**Notes** CORBA compliant.  
**See Also** Other `String_var` constructors.

### **CORBA::String\_var::~~String\_var()**

**Synopsis** `~String_var();`  
**Description** Destructor. The destructor frees the `char*` pointer using `CORBA::string_free()`.  
**Notes** CORBA compliant.

**CORBA::String\_var::operator=()**

**Synopsis**     `String_var& operator=(char* p);`  
              `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& operator[](ULong index);`  
              `char operator[](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**     `operator const 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 373.

If you want to handle these individual system exceptions then the following C++ macro must be defined *before* including the standard file `<CORBA.h>`:

```
#define EXCEPTIONS
```

`CORBA.h` does not normally include the specific definitions of these system exceptions to reduce its size and increase the speed of C++ compilations.

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 C++ Administrator's Guide* 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** `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** `SystemException(ULong minor_id,  
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** `SystemException(const 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 ~SystemException();`
- Description** Destructor.
- Notes** CORBA compliant. It should not be necessary for you to use this destructor.

**CORBA::SystemException::operator=()**

- Synopsis** `const SystemException& operator=(const SystemException&);`
- Description** Assignment operator.
- Notes** CORBA compliant. It should not be necessary for you to use this operator.

### **operator<<()**

- Synopsis** `friend ostream& operator<<(ostream& o, 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** `static SystemException* _narrow(Exception* e);`
- Description** Narrows `Exception` `e` to a `SystemException`. If the runtime type of `e` is not of class `SystemException` or one of its derived classes, `_narrow()` returns a null pointer. Otherwise, `_narrow()` returns a valid `SystemException` pointer. If `e` is a null pointer, `_narrow()` 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** `CompletionStatus completed() const;`
- Description** Returns an indication of the status of an operation at the time the exception was raised. This is one of `COMPLETED_YES`, `COMPLETED_NO`, or `COMPLETED_MAYBE`.
- Notes** CORBA compliant.
- See Also** `CORBA::SystemException::completed()`  
`CORBA::CompletionStatus`



**CORBA::SystemException::completed()**

- Synopsis** `void completed(CompletionStatus completion_status);`
- Description** Sets the status of an operation at the time an exception is raised. This is one of `COMPLETED_YES`, `COMPLETED_NO`, or `COMPLETED_MAYBE`.
- Notes** CORBA compliant. It should not be necessary for you to use this function.
- See Also** `CORBA::SystemException::completed()`  
`CORBA::CompletionStatus`

**CORBA::CompletionStatus**

- Synopsis** `enum CompletionStatus {  
    COMPLETED_YES, COMPLETED_NO, COMPLETED_MAYBE };`
- Description** Enumerates the possible operation completion status values at the time an exception is raised.
- |                              |                                                                                                                                                            |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>COMPLETED_YES</code>   | The requested operation had completed processing prior to the exception being raised.                                                                      |
| <code>COMPLETED_NO</code>    | The requested operation was never initiated.                                                                                                               |
| <code>COMPLETED_MAYBE</code> | 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** `ULong 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 minor(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**

```
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**

Orbis 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           |

| KIND         | PARAMETER LIST                                                                                       |
|--------------|------------------------------------------------------------------------------------------------------|
| tk_any       | NONE                                                                                                 |
| tk_TypeCode  | NONE                                                                                                 |
| tk_Principal | NONE                                                                                                 |
| tk_fixed     | { digits, scale }                                                                                    |
| tk_objref    | { interface-id }                                                                                     |
| tk_struct    | { struct-name,<br>member-name, TypeCode,<br>...<repeat pairs>... }                                   |
| tk_union     | { union-name, switch-TypeCode,<br>label-value, member-name,<br>TypeCode,<br>...<repeat triples>... } |
| tk_enum      | { enum-name, enum-id,<br>...<repeat enum-id>... }                                                    |
| tk_string    | { maxlen-integer }                                                                                   |
| tk_sequence  | { TypeCode, maxlen-integer }                                                                         |
| tk_array     | { TypeCode, length-integer,<br><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,
 Environment& env = default_environment) const;
int operator==(const TypeCode& tc) const;
int operator!=(const TypeCode& tc) const;

TCKind kind(
 Environment& env = default_environment) const;

Long param_count(
 Environment& env = default_environment) const;

Any parameter(Long index,
 Environment& env = default_environment) const;
static TypeCode_ptr IT_create(const char* tc,
 Environment& = default_environment);

static TypeCode_ptr IT_create(const TypeCode_ptr& tc,
 Environment& = default_environment);

static TypeCode_ptr _duplicate(
 TypeCode_ptr,
 Environment &env = default_environment);

static TypeCode_ptr _nil(
 Environment& env = default_environment);
};
```

## CORBA::TypeCode::TypeCode()

- Synopsis** `TypeCode();`
- Description** Default constructor. The `TypeCode` is created with the default kind, `tk_null`.
- Notes** Orbix specific.
- See Also** `CORBA::TypeCode::IT_create()`  
Other `TypeCode` constructor.

### **CORBA::TypeCode::TypeCode()**

**Synopsis** `TypeCode(const TypeCode&);`

**Description** Copy constructor.

**Notes** Orbix specific.

**See Also** `CORBA::TypeCode::IT_create()`  
Other `TypeCode` constructor.

### **CORBA::TypeCode::~~TypeCode()**

**Synopsis** `~TypeCode();`

**Description** Destructor.

**Notes** Orbix specific.

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

**Synopsis** `const TypeCode& operator=(const TypeCode& tc);`

**Description** Assignment operator.

**Notes** Orbix specific.

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

**Synopsis** `int operator==(const 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 equal; returns 0 (false) otherwise.

**Notes** Orbix specific.  
`CORBA::TypeCode::equal()` is the CORBA compliant version of this function.

**See Also** `CORBA::TypeCode::equal()`  
`CORBA::TypeCode::operator!=(())`

**CORBA::TypeCode::operator!=(())**

- Synopsis** `int operator!=(const 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 TypeCode_ptr _duplicate(TypeCode_ptr obj,  
Environment& env = default_environment);`
- 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 TypeCode_ptr _nil(  
Environment& env = default_environment);`
- Description** Returns a nil object reference for a `TypeCode`.
- Notes** CORBA compliant.
- See Also** `CORBA::is_nil( )`

### **CORBA::TypeCode::equal()**

- Synopsis**      `Boolean equal(TypeCode_ptr tc,  
                  Environment& env = default_environment) 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 `true` if the `TypeCodes` are equal; returns `false` otherwise.
- Notes**            CORBA compliant.
- See Also**        `CORBA::TypeCode::operator==( )`  
`CORBA::TypeCode::operator!=( )`

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

- Synopsis**      `static TypeCode_ptr IT_create(const TypeCode_ptr& tc,  
                              Environment& = default_environment);`
- 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 kind(  
                  Environment& env = default_environment) const;`
- Description**    Finds the kind of the `TypeCode`, an enumerated value of type `TCKind`.
- Notes**            CORBA compliant.

**CORBA::TypeCode::param\_count()**

- Synopsis** `Long param_count(  
 Environment& env = default_environment) 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** `Any parameter(Long index,  
 Environment& env = default_environment) 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.

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 C++ 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** `UserCVHandler(const char* identifier);`

**Description** Constructor.

**Parameters**

`identifier`      The name of the configuration value handler.

**Notes**      Orbix specific.

### **CORBA::UserCVHandler::~~UserCVHandler()**

**Synopsis** `virtual ~UserCVHandler();`

**Description** Destructor.

**Notes**      Orbix specific.

### **CORBA::UserCVHandler::GetValue()**

**Synopsis** `virtual unsigned int 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** `UserException();`

**Description** Default constructor.

**Notes** CORBA compliant.

**See Also** Other `UserException` constructors.

### **CORBA::UserException::UserException()**

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <b>Synopsis</b>    | <code>UserException(const UserException&amp;);</code> |
| <b>Description</b> | Copy constructor.                                     |
| <b>Notes</b>       | CORBA compliant.                                      |
| <b>See Also</b>    | Other <code>UserException</code> constructors.        |

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

|                    |                                                                            |
|--------------------|----------------------------------------------------------------------------|
| <b>Synopsis</b>    | <code>const UserException&amp; operator=(const UserException&amp;);</code> |
| <b>Description</b> | Assignment operator.                                                       |
| <b>Notes</b>       | CORBA compliant.                                                           |

### **CORBA::UserException::\_narrow()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>    | <code>static UserException* _narrow(<br/>Exception* e);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Description</b> | <p>Narrows <code>Exception</code> <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> |
| <b>Notes</b>       | 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** `UserOutput(const char* identifier);`

**Description** Constructor.

**Parameters**

`identifier` The name of this output handler.

**Notes** Orbix specific.

### **CORBA::UserOutput::~~UserOutput()**

**Synopsis** `virtual ~UserOutput();`

**Description** Destructor.

**Notes** Orbix specific.

### **CORBA::UserOutput::Output()**

**Synopsis** `virtual void 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.

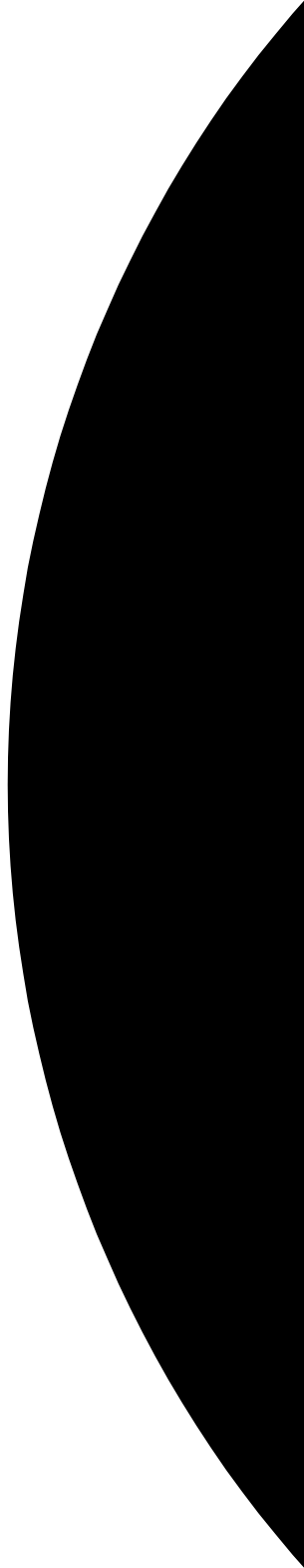
`level` The diagnostic level for this message. (All Orbix diagnostic messages are level 1.)

**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

- Synopsis** `attribute IDLType type_def;`
- Description** Describes the type for this attribute. The same information is contained in the `type` attribute. Changing the `type_def` attribute automatically changes the `type` attribute.
- Notes** CORBA compliant.
- See Also** `CORBA::IDLType`
`CORBA::AttributeDef::type`

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 `IObject`.

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 it's 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</code> ID 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

<code>search_name</code>	The simple name of the object to search for. The use of the wildcard character "*" is also allowed (Orbix specific).
<code>levels_to_search</code>	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.
<code>limit_type</code>	If this is set to <code>dk_all</code> , all the contained Interface Repository objects are returned. If set to the <code>DefinitionKind</code> for a particular Interface Repository kind, it returns only objects of that kind. For example, if set to <code>dk_Operation</code> , it returns contained operations only.
<code>exclude_inherited</code>	Applies only to interfaces. If set to <code>TRUE</code> , inherited objects are not returned. If set to <code>FALSE</code> , 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:

<code>contained_object</code>	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.
<code>kind</code>	The kind of the object being described.
<code>value</code>	An any that may contain one of the following structs:

```
ModuleDescription
ConstantDescription
TypeDescription
ExceptionDescription
AttributeDescription
ParameterDescription
OperationDescription
InterfaceDescription
```

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.

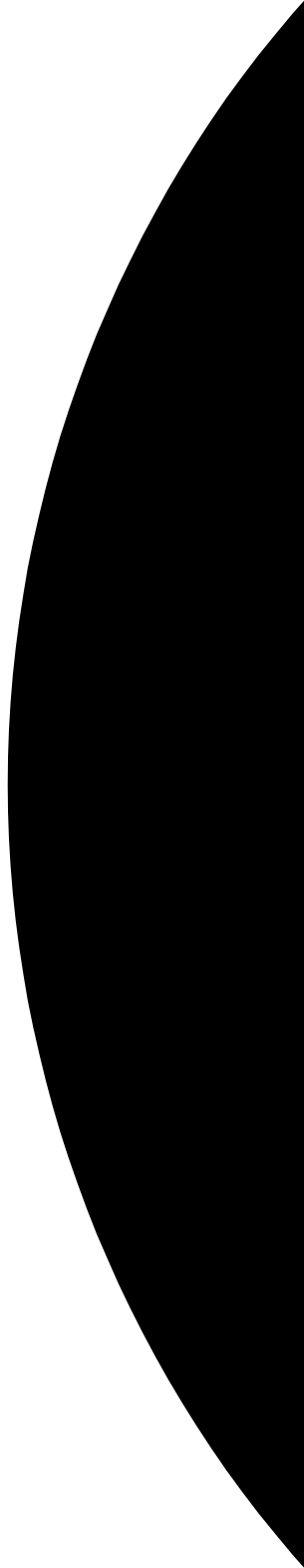
See Also `CORBA::IDLType::type`
`CORBA::UnionDef::dis`

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 itself an Orbix server whose IDL interface is 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 involved, if at all, only with the first operation request from a client—it is not involved with subsequent requests. The Orbix daemon executable is called `orbixd` (`orbixd.exe` on Windows 95 and Windows NT).

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.

The Orbix daemon is also used to search for an appropriate server via the locator and manages the configuration files used by the default locator:

Description	UNIX	Windows 95, Windows NT
The <i>server location</i> file maintains a mapping from a server name to a list of hosts and host groups on which the server runs.	<code>Orbix.hosts</code>	<code>orbix.hst</code>
The <i>host groups definition</i> file defines the host groups.	<code>Orbix.hostgroups</code>	<code>orbix.hgp</code>

Naturally, programs can alternatively edit these files or execute one of the utility commands `serverhosts`, `servergroups` or `grouphosts` as appropriate (see the *Orbix C++ Administrator's Guide*).

As an Orbix server, applications may bind to the Orbix daemon in the normal way. 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("", 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);

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 `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::addGroupsToServer()

Synopsis `boolean addGroupsToServer(
 in string server,
 in stringSeq groupList);`

Description Adds the groups specified in `groupList` to the list of groups for the entry for `server` in the *server location* locator configuration file.

Notes Orbix specific.

See Also `IT_daemon::delGroupsFromServer()`
`CORBA::locatorClass`

IT_daemon::addHostsToGroup()

Synopsis `boolean addHostsToGroup(
 in string group,
 in stringSeq hostList);`

Description Adds the hosts specified in `hostList` to the group, `group`, in the *host groups definition* locator configuration file.

Notes Orbix specific.

See Also `IT_daemon::delHostsFromGroup()`
`CORBA::LocatorClass`

IT_daemon::addHostsToServer()

Synopsis

```
boolean addHostsToServer(  
    in string server,  
    in stringSeq hostList);
```

Description

Adds the host(s) specified in `hostList` to the list of hosts for the entry for `server` in the *server location* locator configuration file.

Notes

Orbix specific.

See Also

`IT_daemon::delHostsFromServer()`
`CORBA::locatorClass`

IT_daemon::addInvokeRights()

Synopsis

```
void 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 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 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 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 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 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 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 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 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 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::delGroupsFromServer()

- Synopsis** `boolean delGroupsFromServer(
 in string server,
 in stringSeq groupList);`
- Description** Deletes the group(s) specified in `groupList` from the list of host groups that support the server `server`. This list is maintained in the *server location* locator configuration file.
- Notes** Orbix specific.
- See Also** `IT_daemon::addGroupsToServer()`

IT_daemon::delHostsFromGroup()

- Synopsis** `boolean delHostsFromGroup(
 in string group,
 in stringSeq hostList);`
- Description** Deletes the hosts specified in `hostList` from the group, `group`, in the *host groups definition* locator configuration file.
- Notes** Orbix specific.
- See Also** `IT_daemon::AddHostsToGroup()`

IT_daemon::delHostsFromServer()

- Synopsis** `boolean delHostsFromServer(
 in string server,
 in stringSeq hostList);`
- Description** Deletes the hosts specified in `hostList` from the list of hosts that support the server `server` in the *server location* locator configuration file.
- Notes** Orbix specific.
- See Also** `IT_daemon::addHostsToServer()`

IT_daemon::getServer()

Synopsis

```
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);
```

Description Gets full information about the Implementation Repository entry for serverName.

Notes Orbix specific.

IT_daemon::killServer()

Synopsis

```
void 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 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 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::listGroupsInServer()

Synopsis

```
boolean listGroupsInServer(
    in string server,
    out stringSeq groupList);
```

Description Returns a list of the host groups (as listed in the *server location* locator configuration file) of which the server `server` is a member.

Notes Orbix specific.

IT_daemon::listHostsInGroup()

Synopsis

```
boolean listHostsInGroup(
    in string group,
    out stringSeq hostList);
```

Description Returns a list of the hosts in the host group, `group`, as listed in the *host groups definition* locator configuration file.

Notes Orbix specific.

IT_daemon::listHostsInServer()

- Synopsis** `boolean listHostsInServer(
 in string server,
 out stringSeq hostList);`
- Description** Returns a list of the hosts on which the server `server` runs as listed in the `server location` configuration file.
- Notes** Orbix specific.

IT_daemon::listServers()

- Synopsis** `void listServers(
 in string subdir,
 out stringSeq servers);`
- Description** Lists all servers in the Implementation Repository directory `subdir`.
- Notes** Orbix specific.

IT_daemon::lookUp()

- Synopsis** `boolean lookUp(
 in string service,
 out stringSeq hostList,
 in octet hops,
 in string tag)`
- Description** Invokes the corresponding `lookUp()` function on the locator. This is normally the default locator—unless an alternative locator has been installed.
- Notes** Orbix specific.
- See Also** `CORBA::locatorClass`
`CORBA::locatorClass::lookUp()`

IT_daemon::newDirectory()

- Synopsis** `void 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 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 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. <ul style="list-style-type: none">0 Multiple-client activation mode.1 Per-client activation mode.2 Per-client-process activation mode.

Notes

Orbix specific.

IT_daemon::newUnSharedServer()

Synopsis

```
void 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. <ul style="list-style-type: none">0 Multiple-client activation mode.1 Per-client activation mode.2 Per-client-process activation mode.

Notes

Orbix specific.

IT_daemon::removeDirRights()

Synopsis

```
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 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 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 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 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 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 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 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 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:

<code>server</code>	The name of the server.
<code>marker</code>	The marker (if any).
<code>principal</code>	The name of the user for whom the server was launched. This is null if the server is not a per-client server.
<code>code</code>	The encoding protocol—for example, XDR.
<code>comms</code>	The transport protocol—for example, TCP/IP.
<code>port</code>	The port used by the communications system.
<code>OSspecific</code>	On UNIX, this is the operating system process identifier of the server process.

`status` One of the enumerated values, `inactive`, `manualLaunch` or `automaticLaunch`.

Notes Orbix specific.

See Also `IT_daemon::LaunchStatus`

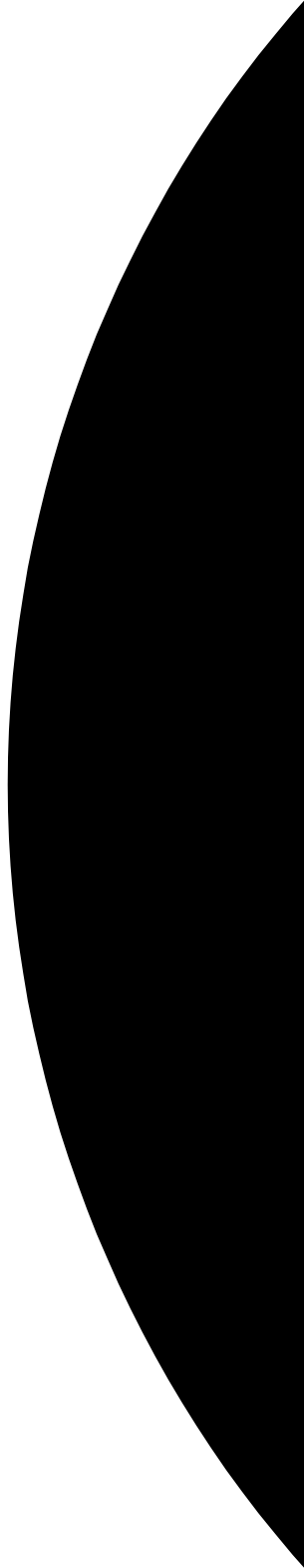
IT_daemon::serverExists()

Synopsis `boolean 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>
    | "::" <identifier>
    | <scoped_name> "::" <identifier>
(12) <const_dcl> ::= "const" <const_type> <identifier>
    | "="<const_exp>
(13) <const_type> ::= <integer_type>
    | <char_type>
    | <boolean_type>
    | <floating_pt_type>
    | <string_type>
    | <scoped_name>
(14) <const_exp> ::= <or_expr>
(15) <or_expr> ::= <xor_expr>
    | <or_expr> "|" <xor_expr>
(16) <xor_expr> ::= <and_expr>
    | <xor_expr> "^" <and_expr>
(17) <and_expr> ::= <shift_expr>
    | <and_expr> "&" <shift_expr>
(18) <shift_expr> ::= <add_expr>
    | <shift_expr> ">>"<add_expr>
    | <shift_expr> "<<"<add_expr>
(19) <add_expr> ::= <mult_expr>
    | <add_expr> "+" <mult_expr>
    | <add_expr> "-" <mult_expr>
(20) <mult_expr> ::= <unary_expr>
    | <mult_expr> "*" <unary_expr>
    | <mult_expr> "/" <unary_expr>
    | <mult_expr> "%" <unary_expr>
(21) <unary_expr> ::= <unary_operator> <primary_expr>
    | <primary_expr>
(22) <unary_operator> ::= "-"
    | "+"
    | "~"

```

(23) <primary_expr>	::= <scoped_name> <literal> "(" <const_exp> ")"
(24) <literal>	::= <integer_literal> <string_literal> <character_literal> <floating_pt_literal> <boolean_literal>
(25) <boolean_literal>	::= "TRUE" "FALSE"
(26) <positive_int_const>	::= <const_exp>
(27) <type_dcl>	::= "typedef" <type_declarator> <struct_type> <union_type> <enum_type>
(28) <type_declarator>	::= <type_spec> <declarators>
(29) <type_spec>	::= <simple_type_spec> <constr_type_spec>
(30) <simple_type_spec>	::= <base_type_spec> <template_type_spec> <scoped_name>
(31) <base_type_spec>	::= <floating_pt_type> <integer_type> <char_type> <boolean_type> <octet_type> <any_type>
(32) <template_type_spec>	::= <sequence_type> <string_type>
(33) <constr_type_spec>	::= <struct_type> <union_type> <enum_type>
(34) <declarators>	::= <declarator> { "," <declarator> }*
(35) <declarator>	::= <simple_declarator>

```

| <complex_declarator>
(36) <simple_declarator> ::= <identifier>
(37) <complex_declarator> ::= <array_declarator>
(38) <floating_pt_type> ::= "float"
| "double"
(39) <integer_type> ::= <signed_int>
| <unsigned_int>
(40) <signed_int> ::= <signed_long_int>
| <signed_short_int>
(41) <signed_long_int> ::= "long"
(42) <signed_short_int> ::= "short"
(43) <unsigned_int> ::= <unsigned_long_int>
| <unsigned_short_int>
(44) <unsigned_long_int> ::= "unsigned" "long"
(45) <unsigned_short_int> ::= "unsigned" "short"
(46) <char_type> ::= "char"
(47) <boolean_type> ::= "boolean"
(48) <octet_type> ::= "octet"
(49) <any_type> ::= "any"
(50) <struct_type> ::= "struct" <identifier>
| "{" <member_list> "}"
(51) <member_list> ::= <member>+
(52) <member> ::= <type_spec> <declarators> ";"
(53) <union_type> ::= "union" <identifier> "switch"
| "(" <switch_type_spec> ")"
| "{" <switch_body> "}"
(54) <switch_type_spec> ::= <integer_type>
| <char_type>
| <boolean_type>
| <enum_type>
| <scoped_name>
(55) <switch_body> ::= <case>+
(56) <case> ::= <case_label>+ <element_spec> ";"
(57) <case_label> ::= "case" <const_exp> ":"

```

		"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
I0000	UNKNOWN	Unknown exception
I0020	BAD_PARAM	An invalid parameter was passed
I0040	NO_MEMORY	Dynamic memory allocation failure
I0060	IMP_LIMIT	Violated implementation limit
I0080	COMM_FAILURE	Communication failure
I0100	INV_OBJREF	Invalid object reference
I0120	NO_PERMISSION	No permission for attempted operation
I0140	INTERNAL	ORB internal error
I0160	MARSHAL	Error marshalling parameter/result
I0180	INITIALIZE	ORB initialisation failure
I0200	NO_IMPLEMENT	Operation implementation unavailable
I0220	BAD_TYPECODE	Bad TypeCode
I0240	BAD_OPERATION	Invalid operation
I0260	NO_RESOURCES	Insufficient resources for request
I0280	NO_RESPONSE	Response to request not yet available
I0300	PERSIST_STORE	Persistent storage failure

Identifier	Exception	Description
I0320	BAD_INV_ORDER	Routine invocations out of order
I0340	TRANSIENT	Transient failure - reissue request
I0360	FREE_MEM	Cannot free memory
I0380	INV_IDENT	Invalid identifier syntax
I0400	INV_FLAG	Invalid flag was specified
I0420	INTF_REPOS	Error accessing interface repository
I0440	BAD_CONTEXT	Error processing context object
I0460	OBJ_ADAPTOR	Failure detected by object adaptor
I0480	DATA_CONVERSION	Data conversion error

System Exceptions Specific to Orbix

Identifier	Orbix Exception	Description
I0500	FILTER_SUPPRESS	Suppress exception raised in per-object pre-filter
I0520	LOCATOR	Locator error
I0540	ASCII_FILE	ASCII file error
I0560	LICENCING	Licensing error
I0580	VXWORKS_EX	VxWorks error
I0600	IIOF	IIOF error
I0620	NO_CONFIG_VALUE	No configuration value set for one of the mandatory configuration entries

Index

Index

A

abortSlowConnects() 167
absolute_name() 288
ActivateCVHandler() 169
ActivateOutputHandler() 169
activationMode 33
active_transactions() 316
add() 131
addDirRights() 347
addForeignFD() 101, 168
addForeignFDSet() 101, 168
addGroupsToServer() 347
addHostsToGroup() 347
addHostsToServer() 348
addInvokeRights() 348
addInvokeRightsDir() 348
add_item() 131
add_item_consume() 132
addLaunchRights() 349
addLaunchRightsDir() 349
addMethod() 349
addSharedMarker() 350
addUnsharedMarker() 350
add_value() 132
add_value_consume() 133
~Any() 19
Any() 18
anyClientsConnected() 33
arg() 8
arguments() 221, 235
assumeArgsOwnership() 221
assumeResultOwnership() 222
_attachPost() 143
_attachPre() 144
AuthenticationFilter() 27

B

baseInterfacesOf() 170
bindUsingIOP() 170
BOA, initialisation 171
BOA_init() 171
bound 329, 331

C

change_implementation() 33
changeOwnerServer() 351
channels, closing 145
clear() 72
_closeChannel() 145
closeChannel() 172
clrf() 96
collocated() 173
commit() 316
completed() 248, 249
CompletionStatus 249
configuration 81
 runtime 79
connectionTimeout() 174
containing_repository() 288
contents() 295
~Context() 57
Context() 56, 57
~ContextIterator() 63
ContextIterator() 63
context_name() 58
contexts 320
continueThreadDispatch() 34
CORBA 182
CORBA 7
CORBA::
 arg() 8
 CompletionStatus 249
 default_environment 9
 extract() 9
 insert() 10
 is_nil() 11
 _LOCATOR_HOPS 7
 _MAX_LOCATOR_HOPS 7
 _OBJECT_TABLE_SIZE_DEFAULT 8
 ORB_init() 12
 release() 13
 string_alloc() 13
 string_dup() 14
 string_free() 14
CORBA::AliasDef 275
CORBA::AliasDef::
 describe() 275

- original_type_def 276
- CORBA::Any 15
- CORBA::Any::
 - ~Any() 19
 - Any() 18
 - operator<<=() 20
 - operator=() 20
 - operator>>=() 22
 - replace() 23
 - type() 24
 - value() 25
- CORBA::ArrayDef 277
- CORBA::ArrayDef::
 - element_type 277
 - element_type_def 278
 - length 278
- CORBA::AttributeDef 279
- CORBA::AttributeDef::
 - describe() 279
 - mode 280
 - type 280
 - type_def 281
- CORBA::BOA 29
- CORBA::BOA::
 - activationMode 33
 - anyClientsConnected() 33
 - change_implementation() 33
 - continueThreadDispatch() 34
 - create() 34
 - deactivate_impl() 35
 - deactivate_obj() 36
 - dispose() 36
 - enableLoaders() 37
 - filterBadConnectAttempts() 37
 - getFileDescriptors() 40
 - getFilter() 40
 - get_id() 38
 - get_principal() 38
 - impl_is_ready() 41
 - isEventPending() 43
 - myActivationMode() 44
 - myImplementationName() 44
 - myImpRepPath() 44
 - myIntRepPath() 45
 - myMarkerName() 45
 - myMarkerPattern() 45
 - myMethodName() 46
 - obj_is_ready() 46
 - processEvents() 47
 - processNextEvent() 48
 - propagateTIEdelete() 49
 - setImpl() 50
 - setNoHangup() 51
- CORBA::ConstantDef 283
- CORBA::ConstantDef::
 - describe() 283
 - type 284
 - type_def 284
 - value 285
- CORBA::Contained 287
- CORBA::Contained::
 - absolute_name() 288
 - containing_repository() 288
 - defined_in 288
 - describe() 289
 - id 290
 - move() 290
 - name() 291
 - version 291
- CORBA::Container 293
- CORBA::Container::
 - contents() 295
 - create_alias() 296
 - create_constant() 296
 - create_enum() 297
 - create_exception() 298
 - create_interface() 299
 - create_module() 299
 - create_struct() 300
 - create_union() 301
 - describe_contents() 302
 - lookup() 302
 - lookup_name() 303
- CORBA::Context 53
- CORBA::Context::
 - ~Context() 57
 - Context() 56, 57
 - context_name() 58
 - create_child() 58
 - delete_values() 59
 - _duplicate() 57
 - get_count() 59
 - get_count_all() 59
 - get_values() 60
 - IT_create() 61
 - _nil() 58
 - parent() 61
 - set_one_value() 62
 - set_values() 62
- CORBA::ContextIterator 63

-
- CORBA::ContextIterator::
 - ~ContextIterator() 63
 - ContextIterator() 63
 - operator>() 64
 - CORBA::DefinitionKind 273
 - CORBA::DynamicImplementation 65
 - CORBA::DynamicImplementation::
 - ~DynamicImplementation() 66
 - DynamicImplementation() 65
 - invoke() 66
 - CORBA::EnumDef 305
 - CORBA::EnumDef::
 - describe() 305
 - members 306
 - CORBA::Environment 67
 - CORBA::Environment::
 - clear() 72
 - _duplicate() 71
 - ~Environment() 70
 - Environment() 69, 70
 - exception() 73
 - int() 74
 - IT_create() 74
 - m_request 75
 - _nil() 72
 - operator 71
 - operator=() 70, 71
 - propagate() 75
 - timeout() 76
 - CORBA::Exception 77
 - CORBA::Exception::
 - ~Exception() 78
 - Exception() 77
 - operator=() 78
 - CORBA::ExceptionDef 307
 - CORBA::ExceptionDef::
 - describe() 307
 - members 308
 - CORBA::ExtraConfigFileCVHandler 79
 - CORBA::ExtraConfigFileCVHandler::
 - ~ExtraConfigFileCVHandler() 80
 - ExtraConfigFileCVHandler() 80
 - CORBA::ExtraRegistryCVHandler 81
 - CORBA::ExtraRegistryCVHandler::
 - ~ExtraRegistryCVHandler() 83
 - ExtraRegistryCVHandler() 82
 - GetRegKey() 83
 - CORBA::Filter 85
 - CORBA::Filter::
 - ~Filter() 86
 - Filter() 86
 - inReplyFailure() 86
 - inReplyPostMarshal() 87
 - inReplyPreMarshal() 88
 - inRequestPostMarshal() 88
 - inRequestPreMarshal() 89
 - outReplyFailure() 90
 - outReplyPostMarshal() 91
 - outReplyPreMarshal() 91
 - outRequestPostMarshal() 92
 - outRequestPreMarshal() 93
 - CORBA::Flags 95
 - CORBA::Flags::
 - clr() 96
 - Flags() 95, 96
 - isNil() 97
 - isSet() 97
 - isSetAll() 97
 - isSetAny() 97
 - operator=() 96
 - reset() 98
 - setArgDef() 98
 - setf() 98
 - ULong() 98
 - CORBA::Identifier 273
 - CORBA::IDLType 311
 - CORBA::IDLType::
 - type 311
 - CORBA::ImplementationDef 99
 - CORBA::ImplementationDef::
 - _duplicate() 99
 - IT_create() 100
 - _nil() 100
 - CORBA::IObject 313
 - CORBA::IObject::
 - def_kind 313
 - destroy() 313
 - CORBA::IT_IOCallback 101
 - CORBA::IT_IOCallback::
 - ForeignFDExcept() 102
 - ForeignFDRead() 102
 - ForeignFDWrite() 102
 - OrbixFDClose() 103
 - OrbixFDOpen() 103
 - CORBA::IT_Repository 315
 - CORBA::IT_Repository::
 - active_transactions() 316
 - commit() 316
 - start() 315
 - CORBA::IT_Repository::rollback() 316

CORBA::IT_reqTransformer 101, 105
CORBA::IT_reqTransformer::
 free_buf() 106
 setRemoteHost() 109
 transform() 107
 transform_error() 108
CORBA::LoaderClass 111
CORBA::LoaderClass::
 load() 113
 ~LoaderClass() 112
 LoaderClass() 112
 record() 114
 rename() 115
 save() 115
CORBA::locatorClass 117
CORBA::locatorClass::
 locatorClass() 117
 lookUp() 118
CORBA::ModuleDef 317
CORBA::ModuleDef::
 describe() 317
CORBA::NamedValue 119
CORBA::NamedValue::
 _duplicate() 121
 flags() 122
 IT_create() 122
 name() 123
 ~NamedValue() 121
 NamedValue() 120
 _nil() 121
 operator=() 121
 value() 123
CORBA::NullLoaderClass 125
CORBA::NullLoaderClass::
 NullLoaderClass() 125
 record() 126
CORBA::NVList 127
CORBA::NVList::
 add() 131
 add_item() 131
 add_item_consume() 132
 add_value() 132
 add_value_consume() 133
 count() 133
 _duplicate() 130
 IT_create() 133
 item() 134
 _nil() 130
 ~NVList() 130
 NVList() 129
 operator=() 130
 remove() 134
CORBA::NVListIterator 135
CORBA::NVListIterator::
 NVListIterator() 135, 136
 operator>() 136
 setList() 136
CORBA::Object 137
CORBA::Object::
 _attachPost() 143
 _attachPre() 144
 _closeChannel() 145
 _create_request() 145
 _deref() 146
 _duplicate() 148
 _enableInternalLock() 148
 _fd() 149
 _get_implementation() 150
 _get_interface() 150
 _getPost() 151
 _getPre() 151
 _hash() 151
 _hasValidOpenChannel() 152
 _host() 152
 _implementation() 153
 _interfaceHost() 153
 _interfaceImplementation() 153
 _interfaceMarker() 154
 _is_a() 154
 _is_equivalent() 155
 _isNull() 155
 _isNullProxy() 156
 _isRemote() 156
 _loader() 157
 _marker() 157
 _nil() 158
 _non_existent() 158
 ~Object() 142
 Object() 140, 141, 142
 _object_to_string() 159
 operator=() 143
 _refCount() 159
 _request() 160
 _save() 160
CORBA::OperationDef 319
CORBA::OperationDef::
 contexts 320
 describe() 320
 exceptions 320
 mode 321

- params 321
- result 322
- result_def 322
- CORBA::ORB 161
- CORBA::ORB:
 - abortSlowConnects() 167
 - ActivateCVHandler() 169
 - ActivateOutputHandler() 169
 - addForeignFD() 101, 168
 - addForeignFDSet() 101, 168
 - baseInterfacesOf() 170
 - bindUsingIOP() 170
 - BOA_init() 171
 - closeChannel() 172
 - collocated() 173
 - connectionTimeout() 174
 - create_environment() 174
 - create_list() 175
 - create_named_value() 175
 - create_operation_list() 176
 - DeactivateCVHandler() 177
 - DeactivateOutputHandler() 177
 - DEFAULT_TIMEOUT 177
 - defaultTxTimeout() 178
 - eagerListeners() 178, 179
 - getAllOrbixFDs() 179
 - GetConfigValue() 182
 - get_default_context() 180
 - getForeignFDSet() 180
 - get_next_response() 181
 - getReqTransformer() 109
 - getSelectableFDSet() 181
 - INFINITE_TIMEOUT 182
 - isBaseInterfaceOf() 183
 - isForeignFD() 183
 - isOrbixFD() 184
 - isOrbixSelectableFD() 184
 - list_initial_services() 185
 - makeIOR() 185
 - makeOrbixObjectKey() 186
 - maxConnectionThreads() 186
 - maxConnectRetries() 187
 - maxFDsPerConnectionThread() 188
 - myHost() 188
 - myServer() 189
 - nativeExceptions() 189
 - noReconnectOnFailure() 190
 - object_to_string() 191
 - optimiseProtocolEncoding() 192
 - Output() 192
 - pingDuringBind() 193
 - PlaceCVHandlerAfter() 193
 - PlaceCVHandlerBefore() 194
 - poll_next_response() 195
 - registerIOCallback() 195
 - registerIOCallbackObject() 197
 - ReinitialiseConfig() 199
 - removeForeignFD() 198
 - removeForeignFDSet() 198
 - reSizeObjectTable() 199
 - resolve_initial_references() 200
 - resortToStatic() 201
 - send_multiple_requests_deferred() 202
 - send_multiple_requests_oneway() 202
 - SetConfigValue() 204
 - setDiagnostics() 205
 - setMyReqTransformer() 109
 - setReqTransformer() 110
 - setServerName() 205
 - set_unsafeDelete() 203
 - string_to_object() 206, 207
 - unregisterIOCallbackObject() 208
 - useHostNameInIOR() 209
 - useTransientPort() 209
- CORBA::PrimitiveDef 323
- CORBA::PrimitiveDef:
 - kind 323
- CORBA::Principal 211
- CORBA::Principal:
 - _duplicate() 212
 - IT_create() 212
 - _nil() 212
 - Principal() 211
- CORBA::Repository 325
- CORBA::Repository:
 - create_array() 326
 - create_sequence() 326
 - create_string() 327
 - describe_contents() 327
 - get_primitive() 327
 - lookup_id() 328
- CORBA::RepositoryId 274
- CORBA::Request 213
- CORBA::Request:
 - arguments() 221, 235
 - assumeOrigArgsOwnership() 221
 - assumeResultOwnership() 222
 - ctx() 222
 - decodeArray() 223
 - descriptor() 224

- `_duplicate()` 220
- `encodeArray()` 224
- `env()` 225
- `extractOctet()` 225
- `get_response()` 225
- `insertOctet()` 226
- `invoke()` 226
- `IT_create()` 227
- `_nil()` 221
- `operation()` 227
- `operator<<()` 219
- `operator>>()` 218
- `poll_response()` 228
- `~Request()` 218
- `Request()` 217
- `reset()` 228
- `result()` 229
- `send_deferred()` 229
- `send_oneway()` 230
- `setOperation()` 230
- `set_return_type()` 231
- `setTarget()` 231
- `target()` 231
- `CORBA::ScopedName` 274
- `CORBA::SequenceDef` 329
- `CORBA::SequenceDef::`
 - `bound` 329
 - `element_type` 330
 - `element_type_def` 330
 - `type` 330
- `CORBA::ServerRequest` 233
- `CORBA::ServerRequest::`
 - `ctx()` 236
 - `env()` 236, 237
 - `op_def()` 237
 - `operation()` 238
 - `op_name()` 238
 - `params()` 239
 - `~Request()` 235
 - `result()` 239
 - `target()` 240
- `CORBA::StringDef` 331
- `CORBA::StringDef::`
 - `bound` 331
- `CORBA::String_var` 241
- `CORBA::String_var::`
 - `char*()` 243
 - `operator=()` 243
 - `~String_var()` 242
 - `String_var()` 242
- `CORBA::StructDef` 333
- `CORBA::StructDef::`
 - `members` 334
- `CORBA::StructDef::describe()` 333
- `CORBA::SystemException` 245
- `CORBA::SystemException::`
 - `completed()` 248, 249
 - `minor()` 249, 250
 - `_narrow()` 248
 - `operator=()` 247
 - `~SystemException()` 247
 - `SystemException()` 246, 247
- `CORBA::ThreadFilter` 251
- `CORBA::ThreadFilter::`
 - `ThreadFilter()` 252
- `CORBA::TypeCode` 253
- `CORBA::TypeCode::`
 - `_duplicate()` 259
 - `equal()` 260
 - `IT_create()` 260
 - `kind()` 260
 - `_nil()` 259
 - `operator!==(())` 259
 - `operator=()` 258
 - `operator==(())` 258
 - `param_count()` 261
 - `parameter()` 261
 - `~TypeCode()` 258
 - `TypeCode()` 257, 258
- `CORBA::TypedefDef` 335
- `CORBA::TypedefDef::`
 - `describe()` 335
- `CORBA::UnionDef` 337
- `CORBA::UnionDef::`
 - `describe()` 337
 - `discriminator_type()` 338
 - `discriminator_type_def()` 338
 - `members` 339
- `CORBA::UserCVHandler` 263
- `CORBA::UserCVHandler::`
 - `GetValue()` 264
 - `~UserCVHandler()` 264
 - `UserCVHandler()` 264
- `CORBA::UserException` 267
- `CORBA::UserException::`
 - `_narrow()` 268
 - `operator =(())` 268
 - `UserException()` 267, 268
- `CORBA::UserOutput` 269
- `CORBA::UserOutput::`

Output() 270
 ~UserOutput() 270
 UserOutput() 270
 CORBA::ExceptionDef::
 type 309
 count() 133
 create() 34
 create_alias() 296
 create_array() 326
 create_child() 58
 create_constant() 296
 create_enum() 297
 create_environment() 174
 create_exception() 298
 create_interface() 299
 create_list() 175
 create_module() 299
 create_named_value() 175
 create_operation_list() 176
 _create_request() 145
 create_sequence() 326
 create_string() 327
 create_struct() 300
 create_union() 301
 ctx() 222, 236

D

Daemon
 IDL definition 343
 DeactivateCVHandler() 177
 deactivate_impl() 35
 deactivate_obj() 36
 DeactivateOutputHandler() 177
 decodeArray() 223
 default_environment 9
 DEFAULT_TIMEOUT 177
 defaultTxTimeout() 178
 defined_in 288
 def_kind 313
 deleteDirectory() 351
 deleteServer() 351
 delete_values() 59
 delGroupsFromServer() 352
 delHostsFromGroup() 352
 delHostsFromServer() 352
 _deref() 146
 describe() 275, 279, 283, 289, 305, 307, 317, 320,
 333, 335, 337
 describe_contents() 302, 327
 descriptor() 224

destroy() 313
 discriminator_type() 338
 discriminator_type_def() 338
 dispose() 36
 _duplicate() 57, 71, 99, 121, 130, 148, 212, 220,
 259
 ~DynamicImplementation() 66
 DynamicImplementation() 65

E

eagerListeners() 178, 179
 element_type 277, 330
 element_type_def 278, 330
 _enableInternalLock() 148
 enableLoaders() 37
 encodeArray() 224
 env() 225, 236, 237
 ~Environment() 70
 Environment() 69, 70
 equal() 260
 ~Exception() 78
 Exception() 77
 exception() 73
 exceptions 320
 ~ExtraConfigFileCVHandler() 80
 ExtraConfigFileCVHandler() 80
 extract() 9
 extractOctet() 225
 ~ExtraRegistryCVHandler() 83
 ExtraRegistryCVHandler() 82

F

_fd() 149
 ~Filter() 86
 Filter() 86
 filterBadConnectAttempts() 37
 Flags() 95, 96
 flags() 122
 free_buf() 106

G

getAllOrbixFDs() 179
 GetConfigValue() 182
 get_count() 59
 get_count_all() 59
 get_default_context() 180
 getFileDescriptors() 40
 getFilter() 40
 getForeignFDSet() 180

get_id() 38
_get_implementation() 150
_get_interface() 150
get_next_response() 181
_getPost() 151
_getPre() 151
get_primitive() 327
get_principal() 38
GetRegKey() 83
getReqTransformer() 109
get_response() 225
getSelectableFDSet() 181
getServer() 353
GetValue() 264
get_values() 60

H

Handlers
 output 269
_hash() 151
_hasValidOpenChannel() 152
_host() 152

I

id 290
IDL definition
 Implementation Repository 343
 Orbix daemon 343
_implementation() 153
impl_is_ready() 41
include files
 daemon.hh 344
INFINITE_TIMEOUT 182
initial services 185
initialisation
 BOA 171
inReplyFailure() 86
inReplyPostMarshal() 87
inReplyPreMarshal() 88
inRequestPostMarshal() 88
inRequestPreMarshal() 89
insert() 10
insertOctet() 226
int() 74
_interfaceHost() 153
_interfaceImplementation() 153
_interfaceMarker() 154
InterfaceName 327
invoke() 66, 226

_is_a() 154
isBaseInterfaceOf() 183
_is_equivalent() 155
isEventPending() 43
isForeignFD() 183
isNil() 97
is_nil() 11
_isNull() 155
_isNullProxy() 156
isOrbixFD() 184
_isRemote() 156
isSet() 97
isSetAll() 97
isSetAny() 97
IT_create() 61, 74, 100, 122, 133, 212, 227, 260
IT_daemon 343
IT_daemon::
 addDirRights() 347
 addGroupsToServer() 347
 addHostsToGroup() 347
 addHostsToServer() 348
 addInvokeRights() 348
 addInvokeRightsDir() 348
 addLaunchRights() 349
 addLaunchRightsDir() 349
 addMethod() 349
 addSharedMarker() 350
 addUnsharedMarker() 350
 changeOwnerServer() 351
 deleteDirectory() 351
 deleteServer() 351
 delGroupsFromServer() 352
 delHostsFromGroup() 352
 delHostsFromServer() 352
 getServer() 353
 killServer() 353
 LaunchStatus 353
 listActiveServers() 354
 listGroupsInServer() 354
 listHostsInGroup() 354
 listHostsInServer() 355
 listServers() 355
 lookup() 355
 newDirectory() 356
 newPerMethodServer() 356
 newSharedServer() 357
 newUnsharedServer() 358
 removeDirRights() 358
 removeInvokeRights() 359
 removeInvokeRightsDir() 359

removeLaunchRights() 359
 removeLaunchRightsDir() 360
 removeMethod() 360
 removeSharedMarker() 360
 removeUnsharedMarker() 361
 serverDetails 361
 serverExists() 362
 item() 134

K

killServer() 353
 kind 323
 kind() 260

L

LaunchStatus 353
 length 278
 listActiveServers() 354
 listGroupsInServer() 354
 listHostsInGroup() 354
 listHostsInServer() 355
 list_initial_services() 185
 listServers() 355
 load() 113
 _loader() 157
 ~LoaderClass() 112
 LoaderClass() 112
 locatorClass() 117
 _LOCATOR_HOPS 7
 lookUp() 118, 355
 lookup() 302
 lookup_id() 328
 lookup_name() 303

M

makeIOR() 185
 makeOrbixObjectKey() 186
 _marker() 157
 maxConnectionThreads() 186
 maxConnectRetries() 187
 maxFDsPerConnectionThread() 188
 _MAX_LOCATOR_HOPS 7
 max_returned_objs 327
 members 306, 308, 334, 339
 minor() 249, 250
 mode 280, 321
 move() 290
 m_request 75
 myActivationMode() 44

myHost() 188
 myImplementationName() 44
 myImpRepPath() 44
 myIntRepPath() 45
 myMarkerName() 45
 myMarkerPattern() 45
 myMethodName() 46
 myServer() 189

N

name() 123, 291
 ~NamedValue() 121
 NamedValue() 120
 _narrow() 248, 268
 nativeExceptions() 189
 newDirectory() 356
 newPerMethodServer() 356
 newSharedServer() 357
 newUnSharedServer() 358
 _nil() 58, 72, 100, 121, 130, 158, 212, 221, 259
 _non_existent() 158
 noReconnectOnFailure() 190
 NullLoaderClass() 125
 ~NVList() 130
 NVList() 129
 NVListIterator() 135, 136

O

~Object() 142
 Object() 140, 141, 142
 _OBJECT_TABLE_SIZE_DEFAULT 8
 _object_to_string() 159
 object_to_string() 191
 obj_is_ready() 46
 op_def() 237
 operation() 227, 238
 operator 71
 operator!==() 259
 operator()() 64, 136
 operator<<() 219, 248
 operator<<=() 20
 operator==() 20, 70, 71, 78, 96, 121, 130, 143, 243,
 247, 258
 operator==() 258
 operator>>() 218
 operator>>=() 22
 op_name() 238
 optimiseProtocolEncoding() 192
 ORB_init() 12

orbixd, IDL definition for 343
original_type_def 276
Output Handlers 269
Output() 192, 270
outReplyFailure() 90
outReplyPostMarshal() 91
outReplyPreMarshal() 91
outRequestPostMarshal() 92
outRequestPreMarshal() 93

P

param_count() 261
parameter() 261
params 321
params() 239
parent() 61
pingDuringBind() 193
PlaceCVHandlerAfter() 193
PlaceCVHandlerBefore() 194
poll_next_response() 195
poll_response() 228
Principal() 211
processEvents() 47
processNextEvent() 48
propagate() 75
propagateTIEdelete() 49

R

record() 114, 126
_refCount() 159
registerIOCallback() 195
registerIOCallbackObject() 197
registry 81
ReinitialiseConfig() 199
release() 13
remove() 134
removeDirRights() 358
removeForeignFD() 198
removeForeignFDSet() 198
removeInvokeRights() 359
removeInvokeRightsDir() 359
removeLaunchRights() 359
removeLaunchRightsDir() 360
removeMethod() 360
removeSharedMarker() 360
removeUnsharedMarker() 361
rename() 115
replace() 23
~Request() 218, 235

Request() 217, 235
_request() 160
reset() 98, 228
reSizeObjectTable() 199
resolve_initial_references() 200
resortToStatic() 201
result 322
result() 229, 239
rollBack() 316

S

_save() 160
save() 115
send_deferred() 229
send_multiple_requests_deferred() 202
send_multiple_requests_oneway() 202
send_oneway() 230
serverDetails 361
serverExists() 362
setArgDef() 98
SetConfigValue() 204
setDiagnostics() 205
setf() 98
setImpl() 50
setList() 136
setMyReqTransformer() 109
setNoHangup() 51
set_one_value() 62
setOperation() 230
setRemoteHost() 109
setReqTransformer() 110
set_return_type() 231
setServerName() 205
setTarget() 231
set_unsafeDelete() 203
set_values() 62
start() 315
string_alloc() 13
string_dup() 14
string_free() 14
string_to_object() 206, 207
~String_var() 242
String_var() 242
system registry 81
~SystemException() 247
SystemException() 246, 247

T

target() 231, 240
ThreadFilter() 252
timeout() 76
transform() 107
transform_error() 108
type 280, 284, 309, 311, 330
type() 24
~TypeCode() 258
TypeCode() 257, 258
type_def 281, 284

U

ULong() 98
unregisterIOCallbackObject() 208
useHostNameInOR() 209
~UserCVHandler() 264
UserCVHandler() 264
UserException() 267, 268
~UserOutput() 270
UserOutput() 270
useTransientPort() 209

V

value 285
value() 25, 123
version 291

