



---

# CORBA Programmer's Reference Java

Version 6.3, December 2005

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

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

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

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

---

#### COPYRIGHT NOTICE

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

Copyright © 2001–2005 IONA Technologies PLC. All rights reserved.

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

Updated: 07-Jun-2006

# Contents

<b>List of Tables</b>	<b>xxv</b>
<b>Preface</b>	<b>xxvii</b>
Audience	xxvii
Organization of this Reference	xxvii
Related Documentation	xxviii
Document Conventions	xxviii
<b>Introduction</b>	<b>1</b>
Interface Repository Quick Reference	2
DII and DSI Quick Reference	4
Value Type Quick Reference	4
About Sequences	5
<b>CORBA Overview</b>	<b>7</b>
<b>Common CORBA Data Types</b>	<b>8</b>
<b>CORBA::AbstractInterfaceDef Interface</b>	<b>27</b>
<b>CORBA::AliasDef Interface</b>	<b>29</b>
<b>CORBA::Any Class</b>	<b>31</b>
<b>CORBA::ArrayDef Interface</b>	<b>41</b>
<b>CORBA::AttributeDef Interface</b>	<b>43</b>
<b>CORBA::ConstantDef Interface</b>	<b>47</b>

<b>CORBA::ConstructionPolicy Interface</b>	<b>51</b>
<b>CORBA::Contained Interface</b>	<b>53</b>
<b>CORBA::Container Interface</b>	<b>59</b>
<b>CORBA::Context Class</b>	<b>79</b>
<b>CORBA::ContextList Class</b>	<b>85</b>
<b>CORBA::Current Interface</b>	<b>87</b>
<b>CORBA::CustomMarshal Value Type</b>	<b>89</b>
<b>CORBA::DataInputStream Value Type</b>	<b>93</b>
<b>CORBA::DataOutputStream Value Type</b>	<b>109</b>
<b>CORBA::DomainManager Interface</b>	<b>125</b>
<b>CORBA::EnumDef Interface</b>	<b>127</b>
<b>CORBA::Environment Class</b>	<b>129</b>
<b>CORBA::Exception Class</b>	<b>131</b>
<b>CORBA::ExceptionDef Interface</b>	<b>133</b>
<b>CORBA::ExceptionList Class</b>	<b>135</b>
<b>CORBA::FixedDef Interface</b>	<b>137</b>
<b>CORBA.InterfaceDefPackage.FullInterfaceDescription Class</b>	<b>139</b>

<b>CORBA::IDLType Interface</b>	<b>141</b>
<b>CORBA::InterfaceDef Interface</b>	<b>143</b>
<b>CORBA::IObject Interface</b>	<b>149</b>
<b>CORBA::ModuleDef Interface</b>	<b>151</b>
<b>CORBA::NamedValue Class</b>	<b>153</b>
<b>CORBA::NativeDef Interface</b>	<b>155</b>
<b>CORBA::NVList Class</b>	<b>157</b>
<b>CORBA::Object Class</b>	<b>161</b>
<b>CORBA::OperationDef Interface</b>	<b>171</b>
<b>CORBA::ORB Class</b>	<b>175</b>
<b>CORBA::Policy Interface</b>	<b>205</b>
Quality of Service Framework	206
Policy Methods	208
<b>CORBA::PolicyCurrent Class</b>	<b>211</b>
<b>CORBA::PolicyManager Class</b>	<b>213</b>
<b>CORBA::PrimitiveDef Interface</b>	<b>215</b>
<b>CORBA::Repository Interface</b>	<b>217</b>
<b>CORBA::Request Class</b>	<b>225</b>

<b>CORBA::SequenceDef Interface</b>	<b>233</b>
<b>CORBA::ServerRequest Class</b>	<b>235</b>
<b>CORBA::String_var Class</b>	<b>241</b>
<b>CORBA::StringDef Interface</b>	<b>245</b>
<b>CORBA::StructDef Interface</b>	<b>247</b>
<b>CORBA::TypeCode Class</b>	<b>249</b>
<b>CORBA::TypedefDef Interface</b>	<b>261</b>
<b>CORBA::UnionDef Interface</b>	<b>263</b>
<b>CORBA::ValueBase Class</b>	<b>265</b>
<b>CORBA::ValueBoxDef Interface</b>	<b>269</b>
<b>CORBA::ValueDef Interface</b>	<b>271</b>
<b>CORBA::ValueFactory</b>	<b>281</b>
CORBA::ValueFactory Type	281
<b>CORBA::ValueMemberDef Interface</b>	<b>283</b>
<b>CORBA::WString_var Class</b>	<b>285</b>
<b>CORBA::WstringDef Interface</b>	<b>289</b>
<b>CosEventChannelAdmin Module</b>	<b>291</b>
CosEventChannelAdmin Exceptions	291

<b>CosEventChannelAdmin::ConsumerAdmin Interface</b>	<b>293</b>
<b>CosEventChannelAdmin::EventChannel Interface</b>	<b>295</b>
<b>CosEventChannelAdmin::ProxyPullConsumer Interface</b>	<b>297</b>
<b>CosEventChannelAdmin::ProxyPullSupplier Interface</b>	<b>299</b>
<b>CosEventChannelAdmin::ProxyPushConsumer Interface</b>	<b>301</b>
<b>CosEventChannelAdmin::ProxyPushSupplier Interface</b>	<b>303</b>
<b>CosEventChannelAdmin::SupplierAdmin Interface</b>	<b>305</b>
<b>CosEventComm Module</b>	<b>307</b>
<b>CosEventComm Exceptions</b>	<b>307</b>
<b>CosEventComm::PullConsumer Interface</b>	<b>309</b>
<b>CosEventComm::PullSupplier Interface</b>	<b>311</b>
<b>CosEventComm::PushConsumer Interface</b>	<b>313</b>
<b>CosEventComm::PushSupplier Interface</b>	<b>315</b>
<b>CosNaming Overview</b>	<b>317</b>
<b>CosNaming::BindingIterator Interface</b>	<b>321</b>
<b>CosNaming::NamingContext Interface</b>	<b>323</b>
<b>CosNaming::NamingContextExt Interface</b>	<b>337</b>

<b>CosNotification Module</b>	<b>341</b>
CosNotification Data Types	341
QoS and Administrative Constant Declarations	342
QoS and Admin Data Types	343
QoS and Admin Exceptions	346
<b>CosNotification::AdminPropertiesAdmin Interface</b>	<b>349</b>
<b>CosNotification::QoSAdmin Interface</b>	<b>351</b>
<b>CosNotifyChannelAdmin Module</b>	<b>355</b>
CosNotifyChannelAdmin Data Types	355
CosNotifyChannelAdmin Exceptions	359
<b>CosNotifyChannelAdmin::ConsumerAdmin Interface</b>	<b>361</b>
<b>CosNotifyChannelAdmin::EventChannel Interface</b>	<b>369</b>
<b>CosNotifyChannelAdmin::EventChannelFactory Interface</b>	<b>375</b>
<b>CosNotifyChannelAdmin::ProxyConsumer Interface</b>	<b>379</b>
<b>CosNotifyChannelAdmin::ProxyPullConsumer Interface</b>	<b>383</b>
<b>CosNotifyChannelAdmin::ProxyPullSupplier Interface</b>	<b>385</b>
<b>CosNotifyChannelAdmin::ProxyPushConsumer Interface</b>	<b>387</b>
<b>CosNotifyChannelAdmin::ProxyPushSupplier Interface</b>	<b>389</b>
<b>CosNotifyChannelAdmin::ProxySupplier Interface</b>	<b>393</b>
<b>CosNotifyChannelAdmin::SequenceProxyPullConsumer Interface</b>	<b>397</b>



<b>CosNotifyChannelAdmin::SequenceProxyPushConsumer Interface</b>	<b>399</b>
<b>CosNotifyChannelAdmin::SequenceProxyPullSupplier Interface</b>	<b>401</b>
<b>CosNotifyChannelAdmin::SequenceProxyPushSupplier Interface</b>	<b>403</b>
<b>CosNotifyChannelAdmin::StructuredProxyPullConsumer Interface</b>	<b>407</b>
<b>CosNotifyChannelAdmin::StructuredProxyPullSupplier Interface</b>	<b>409</b>
<b>CosNotifyChannelAdmin::StructuredProxyPushConsumer Interface</b>	<b>411</b>
<b>CosNotifyChannelAdmin::StructuredProxyPushSupplier Interface</b>	<b>413</b>
<b>CosNotifyChannelAdmin::SupplierAdmin Interface</b>	<b>417</b>
<b>CosNotifyComm Module</b>	<b>425</b>
<b>CosNotifyComm Exceptions</b>	<b>425</b>
<b>CosNotifyComm::NotifyPublish Interface</b>	<b>427</b>
<b>CosNotifyComm::NotifySubscribe Interface</b>	<b>429</b>
<b>CosNotifyComm::PullConsumer Interface</b>	<b>431</b>
<b>CosNotifyComm::PullSupplier Interface</b>	<b>433</b>
<b>CosNotifyComm::PushConsumer Interface</b>	<b>435</b>
<b>CosNotifyComm::PushSupplier Interface</b>	<b>437</b>
<b>CosNotifyComm::SequencePullConsumer Interface</b>	<b>439</b>

<b>CosNotifyComm::SequencePullSupplier Interface</b>	<b>441</b>
<b>CosNotifyComm::SequencePushConsumer Interface</b>	<b>445</b>
<b>CosNotifyComm::SequencePushSupplier Interface</b>	<b>447</b>
<b>CosNotifyComm::StructuredPullConsumer Interface</b>	<b>449</b>
<b>CosNotifyComm::StructuredPullSupplier Interface</b>	<b>451</b>
<b>CosNotifyComm::StructuredPushConsumer Interface</b>	<b>453</b>
<b>CosNotifyComm::StructuredPushSupplier Interface</b>	<b>455</b>
<b>CosNotifyFilter Module</b>	<b>457</b>
CosNotifyFilter Data Types	457
CosNotifyFilter Exceptions	460
<b>CosNotifyFilter::Filter Interface</b>	<b>463</b>
<b>CosNotifyFilter::FilterAdmin Interface</b>	<b>471</b>
<b>CosNotifyFilter::FilterFactory Interface</b>	<b>475</b>
<b>CosNotifyFilter::MappingFilter Interface</b>	<b>477</b>
<b>CosTrading Module</b>	<b>487</b>
CosTrading Data Types	487
CosTrading Exceptions	492
<b>CosTrading::Admin Interface</b>	<b>497</b>
<b>CosTrading::ImportAttributes Interface</b>	<b>505</b>

<b>CosTrading::Link Interface</b>	<b>509</b>
<b>CosTrading::Link Exceptions</b>	<b>510</b>
<b>CosTrading::LinkAttributes Interface</b>	<b>515</b>
<b>CosTrading::Lookup Interface</b>	<b>517</b>
<b>CosTrading::OfferIdIterator Interface</b>	<b>525</b>
<b>CosTrading::OfferIterator Interface</b>	<b>527</b>
<b>CosTrading::Proxy Interface</b>	<b>529</b>
<b>CosTrading::Register Interface</b>	<b>535</b>
<b>CosTrading::SupportAttributes Interface</b>	<b>543</b>
<b>CosTrading::TraderComponents Interface</b>	<b>545</b>
<b>CosTrading::Dynamic Module</b>	<b>547</b>
<b>CosTradingDynamic::DynamicPropEval Interface</b>	<b>549</b>
<b>CosTradingRepos Module</b>	<b>551</b>
<b>CosTradingRepos::ServiceTypeRepository Interface</b>	<b>553</b>
<b>CosTransactions Overview</b>	<b>563</b>
<b>Overview of Classes</b>	<b>563</b>
<b>General Exceptions</b>	<b>564</b>
<b>General Data Types</b>	<b>568</b>
<b>CosTransactions::Control Class</b>	<b>575</b>

<b>CosTransactions::Coordinator Class</b>	<b>577</b>
<b>CosTransactions::Current Class</b>	<b>589</b>
<b>CosTransactions::RecoveryCoordinator Class</b>	<b>595</b>
<b>CosTransactions::Resource Class</b>	<b>597</b>
<b>CosTransactions::SubtransactionAwareResource Class</b>	<b>601</b>
<b>CosTransactions::Synchronization Class</b>	<b>603</b>
<b>CosTransactions::Terminator Class</b>	<b>605</b>
<b>CosTransactions::TransactionalObject Class</b>	<b>607</b>
<b>CosTransactions::TransactionFactory Class</b>	<b>609</b>
<b>CosTypedEventChannelAdmin Module</b>	<b>611</b>
<b>CosTypedEventChannelAdmin Exceptions</b>	<b>611</b>
<b>CosTypedEventChannelAdmin Data Types</b>	<b>612</b>
<b>CosTypedEventChannelAdmin::TypedConsumerAdmin Interface</b>	<b>613</b>
<b>CosTypedEventChannelAdmin::TypedEventChannel Interface</b>	<b>615</b>
<b>CosTypedEventChannelAdmin::TypedProxyPushConsumer Interface</b>	<b>617</b>
<b>CosTypedEventChannelAdmin::TypedSupplierAdmin Interface</b>	<b>619</b>
<b>CosTypedEventComm Module</b>	<b>621</b>
<b>CosTypedEventComm::TypedPushConsumer Interface</b>	<b>623</b>

<b>CSI Overview</b>	<b>625</b>
<b>CSIIOP Overview</b>	<b>629</b>
<b>DsEventLogAdmin Module</b>	<b>635</b>
<b>DsEventLogAdmin::EventLog Interface</b>	<b>637</b>
<b>DsEventLogAdmin::EventLogFactory Interface</b>	<b>639</b>
<b>DsLogAdmin Module</b>	<b>641</b>
DsLogAdmin Exceptions	641
DsLogAdmin Constants	644
DsLogAdmin Datatypes	645
<b>DsLogAdmin::BasicLog Interface</b>	<b>653</b>
<b>DsLogAdmin::BasicLogFactory Interface</b>	<b>655</b>
<b>DsLogAdmin::Iterator Interface</b>	<b>657</b>
<b>DsLogAdmin::Log Interface</b>	<b>659</b>
<b>DsLogAdmin::LogMgr Interface</b>	<b>675</b>
<b>DsLogNotification Module</b>	<b>677</b>
<b>DsNotifyLogAdmin Module</b>	<b>681</b>
<b>DsNotifyLogAdmin::NotifyLog Interface</b>	<b>683</b>
<b>DsNotifyLogAdmin::NotifyLogFactory Interface</b>	<b>685</b>
<b>Dynamic Module</b>	<b>689</b>

<b>DynamicAny Overview</b>	<b>691</b>
<b>DynamicAny::DynAny Class</b>	<b>695</b>
<b>DynamicAny::DynAnyFactory Class</b>	<b>731</b>
<b>DynamicAny::DynArray Class</b>	<b>735</b>
<b>DynamicAny::DynEnum Class</b>	<b>739</b>
<b>DynamicAny::DynFixed Class</b>	<b>743</b>
<b>DynamicAny::DynSequence Class</b>	<b>745</b>
<b>DynamicAny::DynStruct Class</b>	<b>751</b>
<b>DynamicAny::DynUnion Class</b>	<b>757</b>
<b>DynamicAny::DynValue Class</b>	<b>763</b>
<b>GSSUP Overview</b>	<b>769</b>
<b>The IT_Buffer Module</b>	<b>771</b>
IT_Buffer	772
IT_Buffer::Storage	773
IT_Buffer::Segment	775
IT_Buffer::Buffer	777
IT_Buffer::BufferManager	782
<b>IT_Certificate Overview</b>	<b>785</b>
<b>IT_Certificate::AVA Interface</b>	<b>787</b>
<b>IT_Certificate::AVAList Interface</b>	<b>791</b>

<b>IT_Certificate::Certificate Interface</b>	<b>795</b>
<b>IT_Certificate::Extension Interface</b>	<b>797</b>
<b>IT_Certificate::ExtensionList Interface</b>	<b>799</b>
<b>IT_Certificate::X509Cert Interface</b>	<b>803</b>
<b>IT_Certificate::X509CertificateFactory Interface</b>	<b>805</b>
<b>IT_Config Overview</b>	<b>809</b>
<b>IT_Config::Configuration Interface</b>	<b>813</b>
<b>IT_Config::Listener Interface</b>	<b>821</b>
<b>IT_CORBA Overview</b>	<b>827</b>
<b>IT_CORBA::RefCountedLocalObject Class</b>	<b>829</b>
<b>IT_CORBA::RefCountedLocalObjectNC Class</b>	<b>831</b>
<b>IT_CORBA::WellKnownAddressingPolicy Class</b>	<b>833</b>
<b>The IT_CORBASEC Module</b>	<b>835</b>
IT_CORBASEC	836
IT_CORBASEC::ExtendedReceivedCredentials	839
<b>IT_CosTransactions Module</b>	<b>841</b>
<b>IT_CosTransactions::Current Class</b>	<b>843</b>
<b>IT_CSI Overview</b>	<b>845</b>

## CONTENTS

<b>IT_CSI::AttributeServicePolicy Interface</b>	<b>855</b>
<b>IT_CSI::AttributeServiceProtocolClientPolicy Interface</b>	<b>859</b>
<b>IT_CSI::AttributeServiceProtocolServerPolicy Interface</b>	<b>861</b>
<b>IT_CSI::AuthenticateGSSUPCredentials Interface</b>	<b>863</b>
<b>IT_CSI::AuthenticationServicePolicy Interface</b>	<b>869</b>
<b>IT_CSI::CredentialsListObject Interface</b>	<b>875</b>
<b>IT_CSI::CSICredentials Interface</b>	<b>877</b>
<b>IT_CSI::CSICurrent Interface</b>	<b>879</b>
<b>IT_CSI::CSICurrent2 Interface</b>	<b>881</b>
<b>IT_CSI::CSICurrent3 Interface</b>	<b>887</b>
<b>IT_CSI::CSIReceivedCredentials Interface</b>	<b>891</b>
<b>IT_EventChannelAdmin Module</b>	<b>895</b>
IT_EventChannelAdmin Data Types	895
IT_EventChannelAdmin Exceptions	896
<b>IT_EventChannelAdmin::EventChannelFactory Interface</b>	<b>897</b>
<b>IT_FPS Module</b>	<b>901</b>
<b>IT_FPS::InterdictionPolicy Interface</b>	<b>903</b>
<b>The IT_GIOP Module</b>	<b>905</b>
IT_GIOP	906



IT_GIOP::ClientVersionConstraintsPolicy	907
IT_GIOP::ClientCodeSetConstraintsPolicy	908
IT_GIOP::Current	909
IT_GIOP::Current2	913
<b>IT_LoadBalancing Overview</b>	<b>917</b>
<b>IT_LoadBalancing::ObjectGroup Interface</b>	<b>921</b>
<b>IT_LoadBalancing::ObjectGroupFactory Interface</b>	<b>927</b>
<b>IT_Logging Overview</b>	<b>931</b>
<b>IT_Logging::EventLog Interface</b>	<b>939</b>
<b>IT_Logging::LogStream Interface</b>	<b>945</b>
<b>IT_Logging::PrecisionLogStream Interface</b>	<b>949</b>
<b>IT_MessagingAdmin Module</b>	<b>953</b>
<b>IT_MessagingAdmin::Manager Interface</b>	<b>955</b>
<b>The IT_NamedKey Module</b>	<b>957</b>
IT_NamedKey	958
IT_NamedKey::NamedKeyRegistry	959
<b>IT_Naming Module</b>	<b>963</b>
<b>IT_Naming::IT_NamingContextExt Interface</b>	<b>965</b>
<b>IT_NotifyChannelAdmin Module</b>	<b>967</b>
<b>IT_NotifyChannelAdmin::GroupProxyPushSupplier Interface</b>	<b>969</b>

<b>IT_NotifyChannelAdmin:GroupSequenceProxyPushSupplier Interface</b>	<b>973</b>
<b>IT_NotifyChannelAdmin::GroupStructuredProxyPushSupplier Interface</b>	<b>977</b>
<b>IT_NotifyComm Module</b>	<b>981</b>
<b>IT_NotifyComm::GroupNotifyPublish Interface</b>	<b>983</b>
<b>IT_NotifyComm::GroupPushConsumer Interface</b>	<b>985</b>
<b>IT_NotifyComm::GroupSequencePushConsumer Interface</b>	<b>987</b>
<b>IT_NotifyComm::GroupStructuredPushConsumer Interface</b>	<b>989</b>
<b>IT_NotifyLogAdmin Module</b>	<b>991</b>
<b>IT_NotifyLogAdmin::NotifyLog Interface</b>	<b>993</b>
<b>IT_NotifyLogAdmin::NotifyLogFactory Interface</b>	<b>995</b>
<b>The IT_PlainTextKey Module</b>	<b>997</b>
IT_PlainTextKey	998
IT_PlainTextKey::Forwarder	999
<b>IT_PortableServer Overview</b>	<b>1001</b>
<b>IT_PortableServer::DispatchWorkQueuePolicy Interface</b>	<b>1005</b>
<b>IT_PortableServer::ObjectDeactivationPolicy Class</b>	<b>1007</b>
<b>IT_PortableServer::PersistenceModePolicy Class</b>	<b>1009</b>
<b>IT_TLS Overview</b>	<b>1011</b>

<b>IT_TLS::CertValidator Interface</b>	<b>1017</b>
<b>IT_TLS_API Overview</b>	<b>1019</b>
<b>IT_TLS_API::CertConstraintsPolicy Interface</b>	<b>1023</b>
<b>IT_TLS_API::CertValidatorPolicy Interface</b>	<b>1025</b>
<b>IT_TLS_API::MaxChainLengthPolicy Interface</b>	<b>1027</b>
<b>IT_TLS_API::SessionCachingPolicy Interface</b>	<b>1029</b>
<b>IT_TLS_API::TLS Interface</b>	<b>1031</b>
<b>IT_TLS_API::TLSCredentials Interface</b>	<b>1033</b>
<b>IT_TLS_API::TLSReceivedCredentials Interface</b>	<b>1035</b>
<b>IT_TLS_API::TLSTargetCredentials Interface</b>	<b>1037</b>
<b>IT_TLS_API::TrustedCAListPolicy Interface</b>	<b>1039</b>
<b>IT_TypedEventChannelAdmin Module</b>	<b>1041</b>
IT_TypedEventChannelAdmin Data Types	1041
<b>IT_TypedEventChannelAdmin::TypedEventChannelFactory Interface</b>	<b>1043</b>
<b>IT_WorkQueue Module</b>	<b>1047</b>
<b>IT_WorkQueue::AutomaticWorkQueue Interface</b>	<b>1049</b>
<b>IT_WorkQueue::AutomaticWorkQueueFactory Interface</b>	<b>1051</b>

<b>IT_WorkQueue::ManualWorkQueue Interface</b>	<b>1053</b>
<b>IT_WorkQueue::ManualWorkQueueFactory Interface</b>	<b>1055</b>
<b>IT_WorkQueue::WorkItem Interface</b>	<b>1057</b>
<b>IT_WorkQueue::WorkQueue Interface</b>	<b>1059</b>
<b>IT_WorkQueue::WorkQueuePolicy Interface</b>	<b>1063</b>
<b>The IT_ZIOP Module</b>	<b>1065</b>
IT_ZIOP	1066
IT_ZIOP::Compressor	1068
IT_ZIOP::CompressorFactory	1070
IT_ZIOP::CompressionManager	1072
IT_ZIOP::CompressionComponent	1075
IT_ZIOP::CompressionComponentFactory	1076
IT_ZIOP::CompressionEnablingPolicy	1077
IT_ZIOP::CompressorIdPolicy	1078
<b>Messaging Overview</b>	<b>1079</b>
<b>Messaging::ExceptionHandler Value Type</b>	<b>1085</b>
<b>Messaging::RebindPolicy Class</b>	<b>1091</b>
<b>Messaging::ReplyHandler Base Class</b>	<b>1093</b>
<b>Messaging::RoutingPolicy Class</b>	<b>1095</b>
<b>Messaging::SyncScopePolicy Class</b>	<b>1097</b>
<b>OrbixEventsAdmin Module</b>	<b>1099</b>

<b>OrbixEventsAdmin::ChannelManager</b>	<b>1101</b>
<b>PortableInterceptor Module</b>	<b>1105</b>
<b>PortableInterceptor::ClientRequestInfo Interface</b>	<b>1107</b>
<b>PortableInterceptor::ClientRequestInterceptor Interface</b>	<b>1115</b>
<b>PortableInterceptor::Current Interface</b>	<b>1121</b>
<b>PortableInterceptor::Interceptor Interface</b>	<b>1123</b>
<b>PortableInterceptor::IORInfo Interface</b>	<b>1125</b>
<b>PortableInterceptor::IORInterceptor Interface</b>	<b>1129</b>
<b>PortableInterceptor::ORBInitializer Interface</b>	<b>1131</b>
<b>PortableInterceptor::ORBInitInfo Interface</b>	<b>1133</b>
<b>PortableInterceptor::PolicyFactory Interface</b>	<b>1141</b>
<b>PortableInterceptor::RequestInfo Interface</b>	<b>1143</b>
<b>PortableInterceptor::ServerRequestInfo Interface</b>	<b>1151</b>
<b>PortableInterceptor::ServerRequestInterceptor Interface</b>	<b>1155</b>
<b>PortableServer Overview</b>	<b>1161</b>
PortableServer Data Types, Constants, and Exceptions	1162
<b>PortableServer::AdapterActivator Interface</b>	<b>1169</b>

<b>PortableServer::Current Interface</b>	<b>1173</b>
<b>PortableServer::DynamicImplementation Class</b>	<b>1175</b>
<b>PortableServer::IdAssignmentPolicy Interface</b>	<b>1177</b>
<b>PortableServer::IdUniquenessPolicy Interface</b>	<b>1179</b>
<b>PortableServer::ImplicitActivationPolicy Interface</b>	<b>1181</b>
<b>PortableServer::LifespanPolicy Interface</b>	<b>1183</b>
<b>PortableServer::POA Interface</b>	<b>1185</b>
<b>PortableServer::POAManager Interface</b>	<b>1209</b>
<b>PortableServer::RequestProcessingPolicy Interface</b>	<b>1215</b>
<b>PortableServer::ServantActivator Interface</b>	<b>1217</b>
<b>PortableServer::ServantBase</b>	<b>1221</b>
<b>PortableServer::ServantLocator Interface</b>	<b>1223</b>
<b>PortableServer::ServantManager Interface</b>	<b>1227</b>
<b>PortableServer::ServantRetentionPolicy Interface</b>	<b>1229</b>
<b>PortableServer::ThreadPolicy Interface</b>	<b>1231</b>
<b>Security Overview</b>	<b>1233</b>
<b>SecurityLevel1 Overview</b>	<b>1245</b>

<b>SecurityLevel1::Current Interface</b>	<b>1247</b>
<b>SecurityLevel2 Overview</b>	<b>1249</b>
<b>SecurityLevel2::Credentials Interface</b>	<b>1251</b>
<b>SecurityLevel2::Current Interface</b>	<b>1257</b>
<b>SecurityLevel2::EstablishTrustPolicy Interface</b>	<b>1259</b>
<b>SecurityLevel2::InvocationCredentialsPolicy Interface</b>	<b>1261</b>
<b>SecurityLevel2::MechanismPolicy Interface</b>	<b>1263</b>
<b>SecurityLevel2::PrincipalAuthenticator Interface</b>	<b>1265</b>
<b>SecurityLevel2::QOPPolicy Interface</b>	<b>1269</b>
<b>SecurityLevel2::ReceivedCredentials Interface</b>	<b>1271</b>
<b>SecurityLevel2::SecurityManager Interface</b>	<b>1273</b>
<b>SecurityLevel2::TargetCredentials Interface</b>	<b>1277</b>
<b>System Exceptions</b>	<b>1279</b>
<b>Index</b>	<b>1285</b>

## CONTENTS



# List of Tables

Table 1: Interface Repository API	2
Table 2: DII and DSI API	4
Table 3: PolicyErrorCode Constants	18
Table 4: Methods of the Object Class	161
Table 5: Methods and Types of the ORB Class	175
Table 6: Policies	205
Table 7: Operations of the Repository Interface	217
Table 8: OTS Exceptions	565
Table 9: System Exceptions	567
Table 10: Log operational states	651
Table 11: DynAny Methods	695
Table 12: Return Values for DynAny::component_count()	702
Table 13: Default Values When Using create_dyn_any_from_type_code()	733
Table 14: Java Specifying Value Lists in Orbix Configuration	817
Table 15: Java Specifying Value Lists in a Command-Line Parameter	817
Table 16:	819
Table 18: IT_LoadBalancing Common Data Types and Exceptions	917
Table 19: IT_Logging Common Data Types, Methods, and Macros	931
Table 20: Authentication Method Constants and Authentication Structures	1020
Table 21: The Messaging Module	1079
Table 22: ClientRequestInfo Validity	1108
Table 23: PortableServer Common Types	1162
Table 24: Policy Defaults for POAs	1194
Table 25: Corresponding Policies for Servant Managers	1227

LIST OF TABLES

# Preface

Orbix is a software environment for building and integrating distributed object-oriented applications. Orbix is a full implementation of the Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG). Orbix fully supports CORBA version 2.3.

This document is based on the CORBA 2.3 standard with some additional features and Orbix-specific enhancements. If you need help with this or any other IONA products, contact IONA at [support@iona.com](mailto:support@iona.com). Comments on IONA documentation can be sent to [docs-support@iona.com](mailto:docs-support@iona.com).

For the latest online versions of Orbix documentation, see the IONA website:

<http://www.iona.com/docs>

## Audience

The reader is expected to understand the fundamentals of writing a distributed application with Orbix. Familiarity with Java is required.

## Organization of this Reference

This reference presents core-product modules in alphabetical order, disregarding IT\_ prefixes in order to keep together related OMG-compliant and Orbix-proprietary modules. For example, modules `CORBA` and `IT_CORBA` are listed in sequence.

Modules that are specific to a service are also grouped together under the service's name—for example, modules `CosPersistentState`, `IT_PSS`, and `IT_PSS_DB` are listed under Persistent State Service.

---

## Related Documentation

This document is part of a set that comes with the Orbix product. Other books in this set include:

- *Orbix Administrator's Guide*
- *CORBA Programmer's Guide*
- *CORBA Code Generation Toolkit Guide*

## 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, methods, 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, a prompt is not used.
%	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
#	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
>	The notation > represents the DOS, WindowsNT, Windows95, or Windows98 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.



# Introduction

This describes all of the standard programmer's API for CORBA and Orbix. This introduction contains the following topics:

- [“Interface Repository Quick Reference”](#)
- [“DII and DSI Quick Reference”](#)
- [“Value Type Quick Reference”](#)
- [“About Sequences”](#)

The rest of the *CORBA Programmer's Reference* contains the following modules and appendix:

[CORBA](#)

[CosNaming](#)

[CosTransactions](#)

[DynamicAny](#)

[IT\\_Config](#)

[IT\\_CORBA](#)

[IT\\_Logging](#)

[IT\\_PortableServer](#)

[PortableInterceptor](#)

[PortableServer](#)

[“System Exceptions”](#)

---

# Interface Repository Quick Reference

The interface repository (IFR) is the component of Orbix that provides persistent storage of IDL definitions. Programs use the following API to query the IFR at runtime to obtain information about IDL definitions:

**Table 1:** *Interface Repository API*

---

<b>CORBA Structures</b>	<b>CORBA Enumerated Types</b>
<a href="#"><u>AttributeDescription</u></a>	<a href="#"><u>AttributeMode</u></a>
<a href="#"><u>ConstantDescription</u></a>	<a href="#"><u>DefinitionKind</u></a>
<a href="#"><u>ExceptionDescription</u></a>	<a href="#"><u>OperationMode</u></a>
<a href="#"><u>Initializer</u></a>	<a href="#"><u>ParameterMode</u></a>
<a href="#"><u>InterfaceDescription</u></a>	<a href="#"><u>PrimitiveKind</u></a>
<a href="#"><u>ModuleDescription</u></a>	<a href="#"><u>TCKind</u></a>
<a href="#"><u>OperationDescription</u></a>	
<a href="#"><u>ParameterDescription</u></a>	
<a href="#"><u>StructMember</u></a>	
<a href="#"><u>TypeDescription</u></a>	
<a href="#"><u>UnionMember</u></a>	
<a href="#"><u>ValueDescription</u></a>	
<a href="#"><u>ValueMember</u></a>	

---



Table 1: *Interface Repository API*

CORBA Classes and Interfaces	Typecode Methods in CORBA::ORB
<a href="#"><u>AliasDef</u></a>	<a href="#"><u>create_abstract_interface_tc()</u></a>
<a href="#"><u>ArrayDef</u></a>	<a href="#"><u>create_alias_tc()</u></a>
<a href="#"><u>AttributeDef</u></a>	<a href="#"><u>create_array_tc()</u></a>
<a href="#"><u>ConstantDef</u></a>	<a href="#"><u>create_enum_tc()</u></a>
<a href="#"><u>Contained</u></a>	<a href="#"><u>create_exception_tc()</u></a>
<a href="#"><u>Container</u></a>	<a href="#"><u>create_fixed_tc()</u></a>
<a href="#"><u>EnumDef</u></a>	<a href="#"><u>create_interface_tc()</u></a>
<a href="#"><u>ExceptionDef</u></a>	<a href="#"><u>create_native_tc()</u></a>
<a href="#"><u>Environment</u></a>	<a href="#"><u>create_recursive_tc()</u></a>
<a href="#"><u>FixedDef</u></a>	<a href="#"><u>create_sequence_tc()</u></a>
<a href="#"><u>IDLType</u></a>	<a href="#"><u>create_string_tc()</u></a>
<a href="#"><u>InterfaceDef</u></a>	<a href="#"><u>create_struct_tc()</u></a>
<a href="#"><u>IRObject</u></a>	<a href="#"><u>create_union_tc()</u></a>
<a href="#"><u>ModuleDef</u></a>	<a href="#"><u>create_value_box_tc()</u></a>
<a href="#"><u>NativeDef</u></a>	<a href="#"><u>create_value_tc()</u></a>
<a href="#"><u>OperationDef</u></a>	<a href="#"><u>create_wstring_tc()</u></a>
<a href="#"><u>PrimitiveDef</u></a>	
<a href="#"><u>Repository</u></a>	
<a href="#"><u>SequenceDef</u></a>	
<a href="#"><u>StringDef</u></a>	
<a href="#"><u>StructDef</u></a>	
<a href="#"><u>TypeCode</u></a>	
<a href="#"><u>TypedefDef</u></a>	
<a href="#"><u>UnionDef</u></a>	
<a href="#"><u>ValueBoxDef</u></a>	
<a href="#"><u>ValueDef</u></a>	
<a href="#"><u>ValueMemberDef</u></a>	
<a href="#"><u>WstringDef</u></a>	

---

## DII and DSI Quick Reference

The client-side dynamic invocation interface (DII) provides for the dynamic creation and invocation of requests for objects. The server-side counterpart to the DII is the dynamic Skeleton interface (DSI) which dynamically handles object invocations. This dynamic system uses the following data structures, interfaces, and classes:

**Table 2:** *DII and DSI API*

<b>DII Classes</b>	<b>DSI Classes</b>
CORBA::ExceptionList CORBA::Request CORBA::TypeCode	CORBA::ServerRequest PortableServer::DynamicImplementation
<b>Key Data Types</b>	<b>DII-Related Methods</b>
CORBA::Any CORBA::Flags CORBA::NamedValue CORBA::NVList	CORBA::Object::_create_request() CORBA::ORB::create_list() CORBA::ORB::create_operation_list() CORBA::ORB::get_default_context()

## Value Type Quick Reference

A value type is the mechanism by which objects can be passed by value in CORBA operations. Value types use the following data structures, methods, and value types from the CORBA module:

### Types

[ValueFactory](#)

### Value Types and Classes

[CustomMarshal](#)

[DataInputStream](#)

[DataOutputStream](#)

[ValueFactory](#)

[ValueDef](#)

## About Sequences

An IDL sequence maps to a class of the same name. For example, an IDL sequence named *TypeSeq* which is made up of a sequence of *Type* IDL data types, has the class *TypeSeq* implemented.

```
// IDL
typedef sequence<Type> TypeSeq;
```



# CORBA Overview

The CORBA namespace implements the IDL CORBA module. Additional introductory chapters describe the common methods and definitions found in the scope of the CORBA namespace.

- [“Common CORBA Data Types”](#)

All classes or interfaces defined in the CORBA namespace are described in the following alphabetically ordered chapters:

<a href="#">AliasDef</a>	<a href="#">ExceptionDef</a>	<a href="#">Repository</a>
<a href="#">Any</a>	<a href="#">ExceptionList</a>	<a href="#">Request</a>
<a href="#">ArrayDef</a>	<a href="#">FixedDef</a>	<a href="#">SequenceDef</a>
<a href="#">AttributeDef</a>	<a href="#">IDLType</a>	<a href="#">ServerRequest</a>
<a href="#">ConstantDef</a>	<a href="#">InterfaceDef</a>	<a href="#">StringDef</a>
<a href="#">Contained</a>	<a href="#">IObject</a>	<a href="#">StructDef</a>
<a href="#">Container</a>	<a href="#">ModuleDef</a>	<a href="#">TypeCode</a>
<a href="#">Current</a>	<a href="#">NamedValue</a>	<a href="#">TypedefDef</a>
<a href="#">CustomMarshal</a>	<a href="#">NativeDef</a>	<a href="#">UnionDef</a>
<a href="#">DataInputStream</a>	<a href="#">NVList</a>	<a href="#">ValueBoxDef</a>
<a href="#">DataOutputStream</a>	<a href="#">Object</a>	<a href="#">ValueDef</a>
<a href="#">DomainManager</a>	<a href="#">OperationDef</a>	<a href="#">ValueFactory</a>
<a href="#">EnumDef</a>	<a href="#">ORB</a>	<a href="#">ValueMemberDef</a>
<a href="#">Environment</a>	<a href="#">Policy</a>	<a href="#">WstringDef</a>
	<a href="#">PolicyCurrent</a>	
	<a href="#">PolicyManager</a>	
	<a href="#">PrimitiveDef</a>	

Some standard system exceptions are also defined in the CORBA module. However, these exceptions are described in [Appendix A](#).

---

# Common CORBA Data Types

This chapter contains details of all common CORBA data types. The following alphabetically ordered list contains a link to the details of each data type:

<a href="#">AttributeDescription</a>	<a href="#">InvalidPolicies</a>	<a href="#">SetOverrideType</a>
<a href="#">AttributeMode</a>	<a href="#">ModuleDescription</a>	<a href="#">StructMember</a>
<a href="#">ConstantDescription</a>	<a href="#">OperationDescription</a>	<a href="#">TCKind</a>
<a href="#">DefinitionKind</a>	<a href="#">OperationMode</a>	<a href="#">TypeDescription</a>
<a href="#">ExceptionDescription</a>	<a href="#">ParameterDescription</a>	<a href="#">UnionMember</a>
<a href="#">Initializer</a>	<a href="#">ParameterMode</a>	<a href="#">ValueDescription</a>
<a href="#">InterfaceDescription</a>	<a href="#">PolicyError</a>	<a href="#">ValueMember</a>
	<a href="#">PolicyErrorCode</a>	
	<a href="#">PolicyList</a>	
	<a href="#">PolicyType</a>	
	<a href="#">PrimitiveKind</a>	
	<a href="#">RepositoryId</a>	
	<a href="#">RepositoryIdSeq</a>	

## CORBA::AttributeDescription Structure

```
// IDL
struct AttributeDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode type;
    AttributeMode mode;
};
```

The description of an interface attribute in the interface repository.

name	The name of the attribute.
id	The identifier of the attribute.

<code>defined_in</code>	The identifier of the interface in which the attribute is defined.
<code>version</code>	The version of the attribute.
<code>type</code>	The data type of the attribute.
<code>mode</code>	The mode of the attribute.

**See Also** [CORBA::AttributeDef](#)

## **CORBA::AttributeMode Enumeration**

```
// IDL
enum AttributeMode {ATTR_NORMAL, ATTR_READONLY};
```

The mode of an attribute in the interface repository.

`ATTR_NORMAL` Mode is read and write.

`ATTR_READONLY` Mode is read-only.

**See Also** [CORBA::AttributeDef](#)

## **CORBA::ConstantDescription Structure**

```
// IDL
struct ConstantDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode type;
    any value;
};
```

The description of a constant in the interface repository.

`name` The name of the constant.

`id` The identifier of the constant.

`defined_in` The identifier of the interface in which the constant is defined.

---

version	The version of the constant.
type	The data type of the constant.
value	The value of the constant.

**See Also** [CORBA::ConstantDef](#)

## CORBA::DefinitionKind Enumeration

```
// IDL
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,
    dk_Wstring, dk_Fixed,
    dk_Value, dk_ValueBox, dk_ValueMember,
    dk_Native
};
```

Identifies the type of an interface repository object.

Each interface repository object has an attribute ([CORBA::IObject::def\\_kind](#)) of the type [DefinitionKind](#) that records the kind of the IFR object. For example, the `def_kind` attribute of an [InterfaceDef](#) object is `dk_interface`. The enumeration constants `dk_none` and `dk_all` have special meanings when searching for an object in a repository.

**See Also** [CORBA::IObject::def\\_kind](#)  
[CORBA::Contained](#)  
[CORBA::Container](#)

## CORBA::ExceptionDescription

```
// Java
public ExceptionDescription(
    java.lang.String name,
    java.lang.String id,
```



---

```
        java.lang.String defined_in,  
        java.lang.String version,  
        org.omg.CORBA.TypeCode type  
    )
```

The description of an exception in the interface repository.

<code>name</code>	The name of the exception.
<code>id</code>	The identifier of the exception.
<code>defined_in</code>	The identifier of the interface in which the exception is defined.
<code>version</code>	The version of the exception.
<code>type</code>	The data type of the exception.

### **CORBA::Initializer Structure**

```
// IDL  
struct Initializer {  
    StructMemberSeq members;  
    Identifier name;  
};  
// Java  
package org.omg.CORBA;  
public final class Initializer  
    implements org.omg.CORBA.portable.IDLEntity {  
    public org.omg.CORBA.StructMember[] members;  
    public Initializer() {}  
    public Initializer( org.omg.CORBA.StructMember[] members )  
        { this.members = members; }  
}
```

An initializer structure for a sequence in the interface repository.

<code>members</code>	The sequence of structure members.
----------------------	------------------------------------

### **CORBA::InterfaceDescription Structure**

```
// IDL
```

---

```

struct InterfaceDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    RepositoryIdSeq base_interfaces;
    boolean is_abstract;
};

// Java
package org.omg.CORBA;
public final class InterfaceDescription
    implements org.omg.CORBA.portable.IDLEntity
{
    public java.lang.String name;
    public java.lang.String id;
    public java.lang.String defined_in;
    public java.lang.String version;
    public java.lang.String[] base_interfaces;
    public boolean is_abstract;
    public InterfaceDescription() {}
    public InterfaceDescription(
        java.lang.String name,
        java.lang.String id,
        java.lang.String defined_in,
        java.lang.String version,
        java.lang.String[] base_interfaces,
        boolean is_abstract )

```

A description of an interface in the interface repository. This structure is returned by the inherited `describe()` method in the [InterfaceDef](#) interface. The structure members consist of the following:

<code>name</code>	The name of the interface.
<code>id</code>	The identifier of the interface.
<code>defined_in</code>	The identifier of where the interface is defined.
<code>version</code>	The version of the interface.
<code>base_interfaces</code>	The sequence of base interfaces from which this interface is derived.
<code>is_abstract</code>	A true value if the interface is an abstract one, a false value otherwise.

---

See Also [CORBA::InterfaceDef::describe\(\)](#)

## CORBA::InvalidPolicies Exception

```
// IDL
exception InvalidPolicies {
    sequence <unsigned short> indices;
};

// Java
package org.omg.CORBA;
public final class InvalidPolicies
    extends org.omg.CORBA.UserException
{
    public short[] indices;
    public InvalidPolicies()
    {
        super(InvalidPoliciesHelper.id());
    }
    public InvalidPolicies( short[] indices )
    {
        super(InvalidPoliciesHelper.id());
        this.indices = indices;
    }
    public InvalidPolicies( String _reason, short[] indices )
    {
        super(InvalidPoliciesHelper.id() + " " + _reason);
        this.indices = indices;
    }
}
```

This exception is thrown by operations that are passed a bad policy. The indicated policies, although valid in some circumstances, are not valid in conjunction with other policies requested or already overridden at this scope.

## CORBA::ModuleDescription Structure

```
// IDL
struct ModuleDescription {
    Identifier name;
```

---

```

    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
};

// Java
package org.omg.CORBA;
public final class ModuleDescription
    implements org.omg.CORBA.portable.IDLEntity
{
    public java.lang.String name;
    public java.lang.String id;
    public java.lang.String defined_in;
    public java.lang.String version;

    public ModuleDescription() {}
    public ModuleDescription(
        java.lang.String name,
        java.lang.String id,
        java.lang.String defined_in,
        java.lang.String version
    )
    {
        this.name = name;
        this.id = id;
        this.defined_in = defined_in;
        this.version = version;
    }
}

```

The description of an IDL module in the interface repository. The structure members consist of the following:

name	The name of the module.
id	The identifier of the module.
defined_in	The identifier of where the module is defined.
version	The version of the module.

**See Also**     [CORBA::ModuleDef](#)

---

## CORBA::OperationDescription Structure

```
// IDL
struct OperationDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode result;
    OperationMode mode;
    ContextIdSeq contexts;
    ParDescriptionSeq parameters;
    ExcDescriptionSeq exceptions;
};
```

This structure describes an IDL operation in the interface repository. The structure members consist of the following:

<code>name</code>	The name of the IDL operation.
<code>id</code>	The identifier of the IDL operation.
<code>defined_in</code>	The identifier of where the IDL operation is defined.
<code>version</code>	The version of the IDL operation.
<code>result</code>	The <a href="#">TypeCode</a> of the result returned by the defined IDL operation.
<code>mode</code>	Specifies whether the IDL operation's mode is normal ( <a href="#">OP_NORMAL</a> ) or one-way ( <a href="#">OP_ONEWAY</a> ).
<code>contexts</code>	The sequence of context identifiers specified in the context clause of the IDL operation.
<code>parameters</code>	The sequence of structures that give details of each parameter of the IDL operation.
<code>exceptions</code>	The sequence of structures containing details of exceptions specified in the <code>raises</code> clause of the IDL operation.

---

## CORBA::OperationMode Enumeration

```
enum OperationMode {OP_NORMAL, OP_ONEWAY};
```

The mode of an IDL operation in the interface repository. An operation's mode indicates its invocation semantics.

`OP_NORMAL` The IDL operation's invocation mode is normal.

`OP_ONEWAY` The IDL operation's invocation mode is oneway which means the operation is invoked only once with no guarantee that the call is delivered.

## CORBA::ORBid Type

```
// IDL
typedef string ORBid;
```

The name that identifies an ORB. `ORBid` strings uniquely identify each ORB used within the same address space in a multi-ORB application. `ORBid` strings (except the empty string) are not managed by the OMG but are allocated by ORB administrators who must ensure that the names are unambiguous.

## CORBA::ParameterDescription Structure

```
// IDL
struct ParameterDescription {
    Identifier name;
    TypeCode type;
    IDLType type_def;
    ParameterMode mode;
};
```

This structure describes an IDL operation's parameter in the interface repository. The structure members consist of the following:

`name` The name of the parameter.

`type` The [TypeCode](#) of the parameter.

`type_def` Identifies the definition of the type for the parameter.

mode Specifies whether the parameter is an in input, output, or input and output parameter.

## CORBA::ParameterMode Enumeration

```
enum ParameterMode {PARAM_IN, PARAM_OUT, PARAM_INOUT};
```

The mode of an IDL operation's parameter in the interface repository.

PARAM\_IN The parameter is passed as input only.  
PARAM\_OUT The parameter is passed as output only.  
PARAM\_INOUT The parameter is passed as both input and output.

## CORBA::PolicyError Exception

```
// IDL
exception PolicyError {
    PolicyErrorCode reason;
};
```

The `PolicyError` exception is thrown to indicate problems with parameter values passed to `ORB::create_policy()`. Possible reasons are described in the [PolicyErrorCode](#).

### See Also

[CORBA::ORB::create\\_policy\(\)](#)  
[CORBA::PolicyErrorCode](#)

## CORBA::PolicyErrorCode Type

```
typedef short PolicyErrorCode;
```

---

A value representing an error when creating a new [Policy](#). The following constants are defined to represent the reasons a request to create a [Policy](#) might be invalid:

**Table 3:** *PolicyErrorCode Constants*

Constant	Explanation
BAD_POLICY	The requested <a href="#">Policy</a> is not understood by the ORB.
UNSUPPORTED_POLICY	The requested <a href="#">Policy</a> is understood to be valid by the ORB, but is not currently supported.
BAD_POLICY_TYPE	The type of the value requested for the <a href="#">Policy</a> is not valid for that <a href="#">PolicyType</a> .
BAD_POLICY_VALUE	The value requested for the <a href="#">Policy</a> is of a valid type but is not within the valid range for that type.
UNSUPPORTED_POLICY_VALUE	The value requested for the <a href="#">Policy</a> is of a valid type and within the valid range for that type, but this valid value is not currently supported.

**See Also** [CORBA::ORB::create\\_policy\(\)](#)

## CORBA::PolicyList Sequence

A list of [Policy](#) objects. Policies affect an ORB's behavior.

**See Also** [CORBA::Policy](#)  
[CORBA::Object::set\\_policy\\_overrides\(\)](#)  
[PortableServer::POA::POA\\_create\\_POA\(\)](#)

[“About Sequences”](#)



---

## CORBA::PolicyType Type

Defines the type of [Policy](#) object.

The CORBA module defines the following constant `PolicyType`:

```
// IDL
const PolicyType SecConstruction = 11;
```

Other valid constant values for a `PolicyType` are described with the definition of the corresponding [Policy](#) object. There are standard OMG values and IONA-specific values.

### See Also

```
CORBA::Policy
CORBA::PolicyTypeSeq
CORBA::ORB::create\_policy\(\)
CORBA::Object::\_get\_policy\(\)
CORBA::DomainManager::get\_domain\_policy\(\)
// IDL
typedef sequence<PolicyType> PolicyTypeSeq;
```

A sequence of [PolicyType](#) data types.

### See Also

```
CORBA::Object::get\_policy\_overrides\(\)
CORBA::PolicyManager::get\_policy\_overrides\(\)
```

## CORBA::PrimitiveKind Enumeration

```
// IDL
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,
    pk_longlong, pk_ulonglong, pk_longdouble,
    pk_wchar, pk_wstring, pk_value_base
};
typedef PrimitiveKind& PrimitiveKind_out;
```

Indicates the kind of primitive type a [PrimitiveDef](#) object represents in the interface repository.

Most kinds are self explanatory with the exception of the following:

- There are no [PrimitiveDef](#) objects with the kind `pk_null`.
- The kind `pk_string` represents an unbounded string.

- 
- The kind `pk_objref` represents the IDL type [Object](#).

**See Also**

[CORBA::PrimitiveDef](#)  
[CORBA::Repository](#)

## **CORBA::RepositoryId Type**

A string that uniquely identifies, in the interface repository, an IDL module, interface, constant, typedef, exception, attribute, value type, value member, value box, native type, or operation.

The format of `RepositoryId` types is a short format name followed by a colon followed by characters, as follows:

*format\_name:string*

The most common format encountered is the OMG IDL format. For example:

`IDL:Pre/B/C:5.3`

This format contains three components separated by colons:

- |                      |  |
|----------------------|--|
| <code>IDL</code>     | The first component is the format name, <code>IDL</code> .   |
| <code>Pre/B/C</code> | The second component is a list of identifiers separated by <code>'/'</code> characters that uniquely identify a repository item and its scope. These identifiers can contain other characters including underscores ( <code>_</code> ), hyphens ( <code>-</code> ), and dots ( <code>.</code> ). |
| <code>5.3</code>     | The third component contains major and minor version numbers separated by a dot ( <code>.</code> ).  |

**See Also**

[CORBA::Repository::lookup\\_id\(\)](#)

## **CORBA::RepositoryIdSeq Sequence**

A sequence of [RepositoryId](#) strings in the interface repository.

**See Also**

[CORBA::RepositoryId](#)  
“About Sequences”

---

## CORBA::SetOverrideType Enumeration

```
// IDL
enum SetOverrideType {SET_OVERRIDE, ADD_OVERRIDE};
```

The type of override to use in the `set_policy_overrides()` method when setting new policies for an object reference. Possible types consist of:

SET_OVERRIDE	Indicates that new policies are to be associated with an object reference.
ADD_OVERRIDE	Indicates that new policies are to be added to the existing set of policies and overrides for an object reference.

## CORBA::StructMember()

```
// Java
public StructMember(
    java.lang.String name,
    org.omg.CORBA.TypeCode type,
    org.omg.CORBA.IDLType type_def
)
```

This describes an IDL structure member in the interface repository. The structure members consist of the following:

name	The name of the member.
type	The <a href="#">TypeCode</a> for the member.
type_def	Identifies the definition of the type for the member.

## CORBA::TCKind Enumeration

```
// IDL
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,
    tk_value, tk_value_box,
    tk_native,
    tk_abstract_interface
};

```

A `TKKind` value indicates the kind of data type for a [TypeCode](#). A [TypeCode](#) is a value that represent an invocation argument type or attribute type, such as that found in the interface repository or with a dynamic `any` type.

### See Also

```

CORBA::TypeCode::kind()
DynamicAny::DynStruct::current_member_kind()
DynamicAny::DynUnion::discriminator_kind()
DynamicAny::DynUnion::member_kind()
DynamicAny::DynValue::current_member_kind()

```

## CORBA::TypeDescription Structure

```

// IDL
struct TypeDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode type;
};

```

This structure describes an IDL data type in the interface repository. The structure members consist of the following:

<code>name</code>	The name of the data type.
<code>id</code>	The identifier for the data type.
<code>defined_in</code>	The identifier of where the data type is defined.
<code>version</code>	The version of the data type.
<code>type</code>	The <a href="#">TypeCode</a> of the data type.

---

## CORBA::UnionMember Structure

```
// IDL
struct UnionMember {
    Identifier name;
    any label;
    TypeCode type;
    IDLType type_def;
};
```

This structure describes an IDL union member in the interface repository. The structure members consist of the following:

name	The name of the union member.
label	The label of the union member.
type	The <a href="#">TypeCode</a> of the union member.
type_def	The IDL data type of the union member.

## CORBA::ValueDescription Structure

```
// IDL
struct ValueDescription {
    Identifier name;
    RepositoryId id;
    boolean is_abstract;
    boolean is_custom;
    RepositoryId defined_in;
    VersionSpec version;
    RepositoryIdSeq supported_interfaces;
    RepositoryIdSeq abstract_base_values;
    boolean is_truncatable;
    RepositoryId base_value;
};
```

The description of an IDL value type in the interface repository. Value types enable the passing of objects by value rather than just passing by reference. The structure members consist of the following:

name	The name of the value type.
id	The identifier of the value type.

---

<code>is_abstract</code>	True if the value type is abstract. False if the value type is not abstract.
<code>is_custom</code>	True if the value type is custom. False if the value type is not custom.
<code>defined_in</code>	The identifier of where the value type is defined.
<code>version</code>	The version of the value type.
<code>supported_interfaces</code>	
<code>abstract_base_values</code>	
<code>is_truncatable</code>	
<code>base_value</code>	

**See Also** [CORBA::ValueDef::describe\(\)](#)

## CORBA::ValueMember Structure

```
// IDL
struct ValueMember {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode type;
    IDLType type_def;
    Visibility access;
};
```

This structure describes an IDL value type member in the interface repository. The structure members consist of the following:

<code>name</code>	The name of the value type member.
<code>id</code>	The identifier of the value type member.
<code>defined_in</code>	The identifier of where the value type member is defined.
<code>version</code>	The version of the value type member.
<code>type</code>	The <a href="#">TypeCode</a> of the value type member.
<code>type_def</code>	The type definition of the value type member.

---

access

The accessibility of the value type member (public or private).





# CORBA::AbstractInterfaceDef Interface

`AbstractInterfaceDef` describes an abstract IDL interface in the interface repository. It inherits from the [InterfaceDef](#) interface.

```
// IDL
interface AbstractInterfaceDef : InterfaceDef
{
};
```



# CORBA::AliasDef Interface

The `AliasDef` interface describes an IDL typedef that aliases another definition in the interface repository. It is used to represent an IDL `typedef`.

```
// IDL in module CORBA.  
interface AliasDef : TypedefDef {  
    attribute IDLType original_type_def;  
};
```

```
// Java  
package org.omg.CORBA;  
  
public interface AliasDef  
    extends AliasDefOperations,  
        org.omg.CORBA.TypedefDef
```

The following items are described for this interface:

- The [describe\(\)](#) IDL operation
- The [original\\_type\\_def](#) attribute

## See Also

[CORBA::Contained](#)  
[CORBA::Container::create\\_alias\(\)](#)

## AliasDef::describe()

```
// IDL  
Description describe();
```

Inherited from [Contained](#) (which is inherited by [TypedefDef](#)). 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](#).

## See Also

[CORBA::TypedefDef::describe\(\)](#)

---

## AliasDef::original\_type\_def Attribute

```
// IDL
attribute IDLType original_type_def;
```

```
// Java
org.omg.CORBA.IDLType original_type_def();
void original_type_def( org.omg.CORBA.IDLType _val );
```

Identifies the type being aliased. Modifying the `original_type_def` attribute will automatically update the `type` attribute (the `type` attribute is inherited from [TypedefDef](#) which in turn inherits it from [IDLType](#)). Both attributes contain the same information.

### See Also

[CORBA::IDLType::type](#)

# CORBA::Any Class

The class `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) or with CORBA services in general.

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 `op()`. A process receiving an `any` must determine what type of value it stores and then extract the value (using the `TypeCode`). Refer to the *CORBA Programmer's Guide* for more details.

Methods are as follows:

<code>create_input_stream()</code>	<code>extract_TypeCode()</code>	<code>insert_Object()</code>
<code>create_output_stream()</code>	<code>extract_ulong()</code>	<code>insert_octet()</code>
<code>equal()</code>	<code>extract_ulonglong()</code>	<code>insert_short()</code>
<code>extract_any()</code>	<code>extract_ushort()</code>	<code>insert_Streamable()</code>
<code>extract_boolean()</code>	<code>extract_Value()</code>	<code>insert_string()</code>
<code>extract_char()</code>	<code>extract_wchar()</code>	<code>insert_TypeCode()</code>
<code>extract_double()</code>	<code>extract_wstring()</code>	<code>insert_ulong()</code>
<code>extract_fixed()</code>	<code>insert_any()</code>	<code>insert_ulonglong()</code>
<code>extract_float()</code>	<code>insert_boolean()</code>	<code>insert_ushort()</code>
<code>extract_long()</code>	<code>insert_char()</code>	<code>insert_Value()</code>
<code>extract_longlong()</code>	<code>insert_double()</code>	<code>insert_Value()</code>
<code>extract_Object()</code>	<code>insert_fixed()</code>	<code>insert_wchar()</code>
<code>extract_octet()</code>	<code>insert_fixed()</code>	<code>insert_wstring()</code>
<code>extract_short()</code>	<code>insert_float()</code>	<code>read_value()</code>
<code>extract_Streamable()</code>	<code>insert_long()</code>	<code>type()</code>
<code>extract_string()</code>	<code>insert_longlong()</code>	<code>write_value()</code>

```
// Java
```

---

```

package org.omg.CORBA;

abstract public class Any implements
    org.omg.CORBA.portable.IDLEntity {
    abstract public boolean equal(org.omg.CORBA.Any a);

    // type code accessors
    abstract public org.omg.CORBA.TypeCode type();
    abstract public void type(org.omg.CORBA.TypeCode t);

    // read and write values to/from streams
    // throw exception when typecode inconsistent with value
    abstract public void read_value(
        org.omg.CORBA.portable.InputStream is,
        org.omg.CORBA.TypeCode t) throws org.omg.CORBA.MARSHAL;
    abstract public void
        write_value(org.omg.CORBA.portable.OutputStream os);
    abstract public org.omg.CORBA.portable.OutputStream
        create_output_stream();
    abstract public org.omg.CORBA.portable.InputStream
        create_input_stream();
    abstract public short extract_short()
        throws org.omg.CORBA.BAD_OPERATION;
    abstract public void insert_short(short s);
    abstract public int extract_long()
        throws org.omg.CORBA.BAD_OPERATION;
    abstract public void insert_long(int i);
    abstract public long extract_longlong()
        throws org.omg.CORBA.BAD_OPERATION;
    abstract public void insert_longlong(long l);
    abstract public short extract_ushort()
        throws org.omg.CORBA.BAD_OPERATION;
    abstract public void insert_ushort(short s);
    abstract public int extract_ulong()
        throws org.omg.CORBA.BAD_OPERATION;
    abstract public void insert_ulong(int i);
    abstract public long extract_ulonglong()
        throws org.omg.CORBA.BAD_OPERATION;
    abstract public void insert_ulonglong(long l);
    abstract public float extract_float()
        throws org.omg.CORBA.BAD_OPERATION;
    abstract public void insert_float(float f);
    abstract public double extract_double()

```

---

```
        throws org.omg.CORBA.BAD_OPERATION;
abstract public void insert_double(double d);
abstract public boolean extract_boolean()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void insert_boolean(boolean b);
abstract public char extract_char()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void insert_char(char c)
    throws org.omg.CORBA.DATA_CONVERSION;
abstract public char extract_wchar()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void insert_wchar(char c);
abstract public byte extract_octet()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void insert_octet(byte b);
abstract public org.omg.CORBA.Any extract_any()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void insert_any(org.omg.CORBA.Any a);
abstract public org.omg.CORBA.Object extract_Object()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void insert_Object(org.omg.CORBA.Object obj);
abstract public java.io.Serializable extract_Value()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void insert_Value(java.io.Serializable v);
abstract public void insert_Value(
    java.io.Serializable v,
    org.omg.CORBA.TypeCode t)
    throws org.omg.CORBA.MARSHAL;

// throw exception when typecode inconsistent with value
abstract public void insert_Object(
    org.omg.CORBA.Object obj,
    org.omg.CORBA.TypeCode t)
    throws org.omg.CORBA.BAD_PARAM;
abstract public String extract_string()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void insert_string(String s)
    throws org.omg.CORBA.DATA_CONVERSION,
    org.omg.CORBA.MARSHAL;
abstract public String extract_wstring()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void insert_wstring(String s)
    throws org.omg.CORBA.MARSHAL;
```

---

```

// insert and extract typecode
abstract public org.omg.CORBA.TypeCode extract_TypeCode()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void insert_TypeCode(org.omg.CORBA.TypeCode t);

// insert and extract non-primitive IDL types
// BAD_INV_ORDER if any doesn't hold a streamable
public org.omg.CORBA.portable.Streamable extract_Streamable()
    throws org.omg.CORBA.BAD_INV_ORDER {
}
public void insert_Streamable(
    org.omg.CORBA.portable.Streamable s) {
}

// insert and extract fixed
public java.math.BigDecimal extract_fixed() {
    throw org.omg.CORBA.NO_IMPLEMENT();
}
public void insert_fixed(java.math.BigDecimal value) {
}
public void insert_fixed(
    java.math.BigDecimal value,
    org.omg.CORBA.TypeCode type)
    throws org.omg.CORBA.BAD_INV_ORDER {
}
}

```

## Any::create\_input\_stream()

```

abstract public org.omg.CORBA.portable.InputStream
    create_input_stream();

```

This method creates an `org.omg.CORBA.portable.InputStream` object for this `Any`, so that the data contained within the `Any` can be accessed through the `read()` methods defined on `InputStream` rather than the `extract()` methods defined on `Any`.

### Parameters

<code>InputStream</code>	The <code>InputStream</code> representing the <code>Any</code> .
--------------------------	--



---

## **Any:create\_output\_stream()**

```
abstract public org.omg.CORBA.portable.OutputStream  
    create_output_stream();
```

This method creates an `org.omg.CORBA.portable.OutputStream` object for this `Any`. This object allows the `Any` to be populated by calling the `write()` methods declared on `OutputStream` instead of using the `insert()` methods of the `Any`.

### **Parameters**

`OutputStream`            The `OutputStream` representing the `Any`

## **Any::equal()**

```
abstract public boolean equal(org.omg.CORBA.Any a);
```

This method compares the type and value of this `Any` with that of the `Any` passed in as a parameter and returns true if the `Any`s are equal.

### **Parameters**

`a`                        The `Any` to compare against.

## **Any::extract\_type()**

```
abstract public short extract_short()  
    throws org.omg.CORBA.BAD_OPERATION;
```

```
abstract public int extract_long()  
    throws org.omg.CORBA.BAD_OPERATION;
```

```
abstract public long extract_longlong()  
    throws org.omg.CORBA.BAD_OPERATION;
```

```
abstract public short extract_ushort()  
    throws org.omg.CORBA.BAD_OPERATION;
```

```
abstract public int extract_ulong()  
    throws org.omg.CORBA.BAD_OPERATION;
```

```
abstract public long extract_ulonglong()  
    throws org.omg.CORBA.BAD_OPERATION;
```

---

```

abstract public float extract_float()
    throws org.omg.CORBA.BAD_OPERATION;

abstract public double extract_double()
    throws org.omg.CORBA.BAD_OPERATION;

abstract public boolean extract_boolean()
    throws org.omg.CORBA.BAD_OPERATION;

abstract public char extract_char()
    throws org.omg.CORBA.BAD_OPERATION;

abstract public char extract_wchar()
    throws org.omg.CORBA.BAD_OPERATION;

abstract public byte extract_octet()
    throws org.omg.CORBA.BAD_OPERATION;

abstract public org.omg.CORBA.Any extract_any()
    throws org.omg.CORBA.BAD_OPERATION;

abstract public org.omg.CORBA.Object extract_Object()
    throws org.omg.CORBA.BAD_OPERATION;

abstract public java.io.Serializable extract_Value()
    throws org.omg.CORBA.BAD_OPERATION;

abstract public String extract_string()
    throws org.omg.CORBA.BAD_OPERATION;

abstract public String extract_wstring()
    throws org.omg.CORBA.BAD_OPERATION;

abstract public org.omg.CORBA.TypeCode extract_TypeCode()
    throws org.omg.CORBA.BAD_OPERATION;

public org.omg.CORBA.portable.Streamable extract_Streamable()
    throws org.omg.CORBA.BAD_INV_ORDER {
    throw new org.omg.CORBA.NO_IMPLEMENT(); }

public java.math.BigDecimal extract_fixed() {
    throw org.omg.CORBA.NO_IMPLEMENT(); }

```

Extracts the value of the indicated type from the `Any`. You can determine the type of the `Any` using the `org.omg.CORBA.Any.type()` method. You can extract the value using the appropriate extraction method. To extract a user defined type, you can also use the Helper classes, for example:

```

org.omg.CORBA.Any a =          // get the any from somewhere
                             // for example, through the DII,
Object val;                   // from one of the CORBA services.

```

---

```
switch(a.type().kind()){
    case org.omg.CORBA.TCKind._tc_short:
        val = new Short(a.extract_short());
        break;

    //etc. for other basic types

    default :
        if(a.type().equal(AStructHelper.type())){
            val = AStructHelper.extract(a);
        }
        // else some other user defined types
        break;
};
```

You can also obtain the same kind of result by using the class `org.omg.CORBA.portable.InputStream`.

### **Any::insert\_type()**

```
abstract public void insert_short(short s);
abstract public void insert_long(int i);
abstract public void insert_longlong(long l);
abstract public void insert_ushort(short s);
abstract public void insert_ulong(int i);
abstract public void insert_ulonglong(long l);
abstract public void insert_float(float f);
abstract public void insert_double(double d);
abstract public void insert_boolean(boolean b);
abstract public void insert_char(char c)
    throws org.omg.CORBA.DATA_CONVERSION;
```

---

```

abstract public void insert_wchar(char c);
abstract public void insert_octet(byte b);
abstract public void insert_any(org.omg.CORBA.Any a);
abstract public void insert_Object(org.omg.CORBA.Object obj);
abstract public void insert_TypeCode(org.omg.CORBA.TypeCode t);
abstract public void insert_Value(java.io.Serializable v);
abstract public void insert_Value(
    java.io.Serializable v,
    org.omg.CORBA.TypeCode t
) throws org.omg.CORBA.MARSHAL;
abstract public void insert_Object(
    org.omg.CORBA.Object obj,
    org.omg.CORBA.TypeCode t
) throws org.omg.CORBA.BAD_PARAM;
abstract public void insert_string(String s)
    throws
        org.omg.CORBA.DATA_CONVERSION,
        org.omg.CORBA.MARSHAL;
abstract public void insert_wstring(String s)
    throws org.omg.CORBA.MARSHAL;
public void insert_fixed(java.math.BigDecimal value)
{ throw new org.omg.CORBA.NO_IMPLEMENT(); }
public void insert_fixed(
    java.math.BigDecimal value,
    org.omg.CORBA.TypeCode type
)
    throws org.omg.CORBA.BAD_INV_ORDER {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
public void insert_Streamable(
    org.omg.CORBA.portable.Streamable s) {
    throw new org.omg.CORBA.NO_IMPLEMENT(); }

```

Insert a value of the indicated type into the `Any`. Previous values held in the `Any` are discarded and each insertion method takes a copy of the value inserted.

---

## Parameters

first parameter      The actual value to insert into the `Any`.  
tc                    The `TypeCode` of the value being inserted.

You can use the `nameHelper` class to insert a user-defined type. For example, given the following IDL:

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

Use the `insert()` method generated on the `AStructHelper` class:

```
//Java
org.omg.CORBA.Any a = new org.omg.CORBA.Any();

AStruct s = new AStruct("String",1.0f);
try {
    AStructHelper.insert(a,s);
}
catch(org.omg.CORBA.SystemException){
    //do something here
}
```

The same result can be achieved using the `OutputStream`.

### **Any::read\_value()**

```
abstract public void read_value(
    org.omg.CORBA.portable.InputStream is,
    org.omg.CORBA.TypeCode t
) throws org.omg.CORBA.MARSHAL;
```

Reads an object from an `InputStream` for the current `Any`.

## Parameters

is                    The `InputStream` to read the data from.  
t                    The `TypeCode` of the object to be read from the stream.

---

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

```
abstract public org.omg.CORBA.TypeCode type();
```

Returns the `TypeCode` of the Object encapsulated within the `Any`.

```
abstract public void type(org.omg.CORBA.TypeCode t);
```

Sets the `TypeCode` of the Object encapsulated within the `Any`.

### **Parameters**

`t`                      The `TypeCode` of the object.

## **Any::write\_value()**

```
abstract public void write_value(  
    org.omg.CORBA.portable.OutputStream os  
);
```

Writes the object contained within the `Any` into the specified `OutputStream`.

### **Parameters**

`os`                      The `OutputStream` to write the data to.

# CORBA::ArrayDef Interface

The `ArrayDef` interface represents a one-dimensional array in an interface repository. 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. An instance of interface `ArrayDef` can be created using [create\\_array\(\)](#).

```
// IDL in module CORBA.  
interface ArrayDef : IDLType {  
    attribute unsigned long length;  
    readonly attribute TypeCode element\_type;  
    attribute IDLType element\_type\_def;  
};
```

## See Also

[CORBA::IDLType](#)  
[CORBA::ArrayDef::element\\_type\\_def](#)  
[CORBA::Repository::create\\_array\(\)](#)

## ArrayDef::element\_type Attribute

```
// IDL  
readonly attribute TypeCode element_type;  
  
// Java  
org.omg.CORBA.TypeCode element_type();
```

Identifies the type of the element contained in the array. This contains the same information as in the `element_type_def` attribute.

## See Also

[CORBA::ArrayDef::element\\_type\\_def](#)

## ArrayDef::element\_type\_def Attribute

```
// IDL  
attribute IDLType element_type_def;
```

---

```
// Java
org.omg.CORBA.IDLType element_type_def();
```

Describes the type of the element contained within the array. This contains the same information as in the attribute `element_type` attribute.

The type of elements contained in the array can be changed by changing this attribute. Changing this attribute also changes the `element_type` attribute.

**See Also**

[CORBA::ArrayDef::element\\_type](#)

### **ArrayDef::length Attribute**

```
// IDL
attribute unsigned long length;
```

```
// Java
int length();
```

Returns the number of elements in the array.

```
void length(int _val);
```

Specifies the number of elements in the array.



# CORBA::AttributeDef Interface

The `AttributeDef` interface describes an attribute of an interface in the interface repository.

```
// IDL in module CORBA.  
interface AttributeDef : Contained {  
    readonly attribute TypeCode type;  
    attribute IDLType type\_def;  
    attribute AttributeMode mode;  
};
```

The inherited [describe\(\)](#) method is also described.

## See Also

[CORBA::Contained](#)  
[CORBA::InterfaceDef::create\\_attribute\(\)](#)

## AttributeDef::describe()

```
// IDL  
Description describe();
```

Inherited from [Contained](#). The [DefinitionKind](#) for the `kind` member of this structure is `dk_Attribute`. The `value` member is an `any` whose [TypeCode](#) is `_tc_AttributeDescription`. The value is a structure of type `AttributeDescription`.

## See Also

[CORBA::Contained::describe\(\)](#)

## AttributeDef::mode Attribute

```
// IDL  
attribute AttributeMode mode;
```

```
// Java  
org.omg.CORBA.AttributeMode mode();
```

Returns the mode of the attribute.

---

```
// Java
void mode(
    org.omg.CORBA.AttributeMode _val
);
```

Specifies whether the attribute is read and write ([ATTR\\_NORMAL](#)) or read-only ([ATTR\\_READONLY](#)).

### AttributeDef::type Attribute

```
// IDL
readonly attribute TypeCode type;
```

```
// Java
org.omg.CORBA.TypeCode type();
```

Returns the type of this attribute. The same information is contained in the `type_def` attribute.

#### See Also

[CORBA::TypeCode](#)  
[CORBA::AttributeDef::type\\_def](#)

### AttributeDef::type\_def Attribute

```
// IDL
attribute IDLType type_def;
```

```
// Java
org.omg.CORBA.IDLType type_def();
```

Returns the type of this attribute.

```
// Java
void type_def(
    org.omg.CORBA.IDLType _val
);
```

---

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.

**See Also**

[CORBA::IDLType](#)

[CORBA::AttributeDef::type](#)



# CORBA::ConstantDef Interface

Interface `ConstantDef` describes an IDL constant in the interface repository. The name of the constant is inherited from `Contained`.

```
// IDL
// in module CORBA.
interface ConstantDef : Contained {
    readonly attribute TypeCode type;
    attribute IDLType type\_def;
    attribute any value;
};

// Java
public interface ConstantDef
    extends ConstantDefOperations,
        org.omg.CORBA.Contained
{
}
```

The inherited operation [describe\(\)](#) is also described.

## See Also

[CORBA::Contained](#)  
[CORBA::Container::create\\_constant\(\)](#)

## ConstantDef::describe()

```
// IDL
Description describe();
```

Inherited from [Contained](#), `describe()` returns a structure of type [Contained::Description](#).

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](#).

## See Also

[CORBA::Contained::describe\(\)](#)

### ConstantDef::type Attribute

```
// IDL
readonly attribute TypeCode type;

// Java
org.omg.CORBA.TypeCode type();
```

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.

#### See Also

[CORBA::ConstantDef::type\\_def](#)

### ConstantDef::type\_def Attribute

```
// IDL
attribute IDLType type_def;

// Java
org.omg.CORBA.IDLType type_def();
```

Returns the type of this constant.

```
void type_def(org.omg.CORBA.IDLType _val);
```

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.

#### See Also

[CORBA::ConstantDef::type](#)

### ConstantDef::value Attribute

```
// IDL
attribute any value;

// Java
org.omg.CORBA.Any value();
```

Returns the value of this attribute.

```
void value(org.omg.CORBA.Any _val);
```

---

Contains the value for this constant. When changing the `value` attribute, the [TypeCode](#) of the `any` must be the same as the `type` attribute.

**See Also**

[CORBA::TypeCode](#)





# CORBA::ConstructionPolicy Interface

When new object references are created, the `ConstructionPolicy` object allows the caller to specify that the instance should be automatically assigned membership in a newly created policy domain. When a policy domain is created, it also has a [DomainManager](#) object associated with it. The `ConstructionPolicy` object provides a single operation that makes the [DomainManager](#) object.

```
// IDL in CORBA Module
interface ConstructionPolicy: Policy {
    void make\_domain\_manager(
        in CORBA::InterfaceDef object_type,
        in boolean constr_policy
    );
};
```

## ConstructionPolicy::make\_domain\_manager()

```
// IDL
void make\_domain\_manager(
    in CORBA::InterfaceDef object_type,
    in boolean constr_policy
);
// Java
void make\_domain\_manager(
    org.omg.CORBA.InterfaceDef object_type,
    boolean constr_policy
);
```

This operation sets the construction policy that is to be in effect in the policy domain for which this `ConstructionPolicy` object is associated.

---

## Parameters

- `object_type` The type of the objects for which domain managers will be created. If this is nil, the policy applies to all objects in the policy domain.
- `constr_policy` A value of true indicates to the ORB that new object references of the specified `object_type` are to be associated with their own separate policy domains (and associated domain manager). Once such a construction policy is set, it can be reversed by invoking [make\\_domain\\_manager\(\)](#) again with the value of false.
- A value of false indicates the construction policy is set to associate the newly created object with the policy domain of the creator or a default policy domain.

You can obtain a reference to the newly created domain manager by calling [\\_get\\_domain\\_managers\(\)](#) on the newly created object reference.

## See Also

[CORBA::DomainManager](#)  
[CORBA::Object::\\_get\\_domain\\_managers\(\)](#)

# CORBA::Contained Interface

Interface `Contained` is an abstract interface that describes 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](#)
- [ValueDef](#)

The complete interface is shown here:

```
// IDL
// In module CORBA.
interface Contained : IRObject {

    // read/write interface
    attribute RepositoryId id;
    attribute Identifier name;
    attribute VersionSpec version;

    // read interface
    readonly attribute Container defined\_in;
    readonly attribute ScopedName absolute\_name;
    readonly attribute Repository containing\_repository;
    struct Description {
        DefinitionKind kind;
        any value;
    };
    Description describe();
}
```

---

```
    // write interface
    void move(
        in Container new_container,
        in Identifier new_name,
        in VersionSpec new_version
    );
};
```

## See Also

[CORBA::Container](#)  
[CORBA::IObject](#)

## Contained::absolute\_name Attribute

```
//IDL
readonly attribute ScopedName absolute_name;

// Java
java.lang.String absolute_name();
```

Gives the absolute scoped name of an object.

## Contained::containing\_repository Attribute

```
// IDL
readonly attribute Repository containing_repository;

// Java
org.omg.CORBA.Repository containing_repository();
```

Gives the [Repository](#) within which the object is contained.

## Contained::defined\_in Attribute

```
// IDL
attribute Container defined_in;

// Java
org.omg.CORBA.Container defined_in();
```

Specifies the Container 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 that inheritance of operations and attributes across the boundaries of different modules is also allowed.

**See Also** [CORBA::Container::contents\(\)](#)

### Contained::describe()

```
// IDL
Description describe();

// Java
org.omg.CORBA.ContainedPackage.Description describe();
```

Returns a structure of type `Description`.

The `kind` field of the `Description` structure contains the same value as the [def\\_kind](#) attribute that [Contained](#) inherits from [IRObject](#).

**See Also** [CORBA::Container::describe\\_contents\(\)](#)  
[CORBA::DefinitionKind](#)

### Contained::Description Structure

```
// IDL
struct Description {
    DefinitionKind kind;
    any value;
};
```

This is a generic form of description which is used as a wrapper for another structure stored in the `value` field.

Depending on the type of the `Contained` object, the `value` field will contain 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).

## Contained::id Attribute

```
// IDL
attribute RepositoryId id;

// Java
java.lang.String id();
void id(java.lang.String _val);
```

A `RepositoryId` provides an alternative method of naming an object.

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](#).

## Contained::move()

```
// IDL
void move(
    in Container new_container,
    in Identifier new_name,
    in VersionSpec new_version
);

// Java
void move(
    org.omg.CORBA.Container new_container,
    java.lang.String new_name,
    java.lang.String new_version
);
```

---

Removes this object from its container, and adds it to the container specified by `new_container`. The new container must:

- Be in the same repository.
- Be capable of containing an object of this type.
- Not contain an object of the same name (unless multiple versions are supported).

The `name` attribute of the object being moved is changed to that specified by the `new_name` parameter. The `version` attribute is changed to that specified by the `new_version` parameter.

**See Also**

[CORBA::Container](#)

## Contained::name Attribute

```
// IDL
attribute Identifier name;

// Java
java.lang.String name();
void name(java.lang.String _val);
```

Return or set 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. The `name` attribute can be changed but it is an error to change it to a value that is currently in use within the object's [Container](#).

**See Also**

[CORBA::Contained::id](#)

## Contained::version Attribute

```
// IDL
attribute VersionSpec version;
```

---

```
// Java
java.lang.String version();
void version(java.lang.String _val);
```

Return or set 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.



# CORBA::Container Interface

Interface `Container` describes objects that can contain other objects in the interface repository. A `Container` can contain any number of objects derived from the [Contained](#) interface. Such objects include:

[AttributeDef](#)  
[ConstantDef](#)  
[ExceptionDef](#)  
[InterfaceDef](#)  
[ModuleDef](#)  
[OperationDef](#)  
[TypedefDef](#)  
[ValueDef](#)  
[ValueMemberDef](#)

The interface is shown here:

```
//IDL
// In CORBA Module
interface Container : IRObjct {
    // read interface
    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
    );

    DescriptionSeq describe_contents(
        in DefinitionKind limit_type,
```

---

```
        in boolean exclude_inherited,
        in long max_returned_objs
    );

    // write interface
    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
    in boolean is_abstract
);

ValueDef create\_value(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in boolean is_custom,
    in boolean is_abstract,
    in ValueDef base_value,
    in boolean is_truncatable,
    in ValueDefSeq abstract_base_values,
    in InterfaceDef supported_interface,
    in InitializerSeq initializers
);

ValueBoxDef create\_value\_box(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType original_type_def
);

ExceptionDef create\_exception(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in StructMemberSeq members
);

NativeDef create\_native(
    in RepositoryId id,
    in Identifier name,
```

---

```
        in VersionSpec version,
    );
}; // End Interface Container
```

**See Also** [CORBA::IObject](#)

## Container::contents()

```
// IDL
ContainedSeq contents(
    in DefinitionKind limit_type,
    in boolean exclude_inherited
);

// Java
org.omg.CORBA.Contained[] contents(
    org.omg.CORBA.DefinitionKind limit_type,
    boolean exclude_inherited
);
```

Returns a sequence of [Contained](#) objects that are directly contained in (defined in or inherited into) the target object. This operation can be used to navigate through the hierarchy of definitions—starting, for example, at a [Repository](#).

### Parameters

<code>limit_type</code>	If set to <code>dk_all</code> , all of the contained interface repository objects are returned. If set to the <a href="#">DefinitionKind</a> for a specific interface type, it returns only interfaces of that type. For example, if set to, <code>dk_Operation</code> , then it returns contained operations only.
<code>exclude_inherited</code>	Applies only to interfaces. If true, no inherited objects are returned. If false, objects are returned even if they are inherited.

**See Also** [CORBA::Container::describe\\_contents\(\)](#)  
[CORBA::DefinitionKind](#)

---

## Container::create\_alias()

```
// IDL
AliasDef create_alias(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType original_type
);

// Java
org.omg.CORBA.AliasDef create_alias(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    org.omg.CORBA.IDLType original_type
);
```

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 repository ID for the new <a href="#">AliasDef</a> object. An exception is raised if an interface repository object with the same ID already exists within the object's repository.
<code>name</code>	The name for the new <a href="#">AliasDef</a> object. It is an error to 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 <a href="#">AliasDef</a> .
<code>original_type</code>	The original type that is being aliased.

### Exceptions

<code>BAD_PARAM,</code> minor code 2	An object with the specified <code>id</code> already exists in the repository.
<code>BAD_PARAM,</code> minor code 3	The specified <code>name</code> already exists within this <code>Container</code> and multiple versions are not supported.
<code>BAD_PARAM,</code> minor code 4	The created object is not allowed by the <code>Container</code> . Certain interfaces derived from <code>Container</code> may restrict the types of definitions that they may contain.

---

**See Also**

[CORBA::AliasDef](#)

## Container::create\_constant()

```
// IDL
ConstantDef create_constant(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType type,
    in any value
);

// Java
org.omg.CORBA.ConstantDef create_constant(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    org.omg.CORBA.IDLType type,
    org.omg.CORBA.Any value
);
```

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

id	The repository ID of the new <a href="#">ConstantDef</a> object. It is an error to specify an ID that already exists within the object's repository.
name	The name of the new <a href="#">ConstantDef</a> object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.
version	The version number of the new <a href="#">ConstantDef</a> object.
type	The type of the defined constant. This must be one of the simple types (long, short, ulong, ushort, float, double, char, string, boolean).
value	The value of the defined constant.

---

## Exceptions

- `BAD_PARAM`, minor code 2 An object with the specified `id` already exists in the repository.
- `BAD_PARAM`, minor code 3 The specified `name` already exists within this `Container` and multiple versions are not supported.
- `BAD_PARAM`, minor code 4 The created object is not allowed by the `Container`. Certain interfaces derived from `Container` may restrict the types of definitions that they may contain.

## See Also

[CORBA::ConstantDef](#)

## Container::create\_enum()

```
// IDL
EnumDef create_enum(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in EnumMemberSeq members
);

// Java
org.omg.CORBA.EnumDef create_enum(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    java.lang.String[] members
);
```

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

- `id` The repository ID of the new [EnumDef](#) object. It is an error to specify an ID that already exists within the [Repository](#).
- `name` The name of the [EnumDef](#) object. It is an error to specify a name that already exists within the object's `Container` when multiple versions are not supported.

---

**version** The version number of the new [EnumDef](#) object.

**members** A sequence of structures that describes the members of the new [EnumDef](#) object.

## Exceptions

**BAD\_PARAM,** minor code 2 An object with the specified `id` already exists in the repository.

**BAD\_PARAM,** minor code 3 The specified `name` already exists within this `Container` and multiple versions are not supported.

**BAD\_PARAM,** minor code 4 The created object is not allowed by the `Container`. Certain interfaces derived from `Container` may restrict the types of definitions that they may contain.

## See Also

[CORBA::EnumDef](#)

## Container::create\_exception()

```
// IDL
ExceptionDef create_exception(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in StructMemberSeq members
);

// Java
org.omg.CORBA.ExceptionDef create_exception(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    org.omg.CORBA.StructMember[] members
);
```

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.



---

## Parameters

<code>id</code>	The repository ID of the new <a href="#">ExceptionDef</a> object. It is an error to specify an ID that already exists within the object's repository.
<code>name</code>	The name of the new <a href="#">ExceptionDef</a> object. It is an error to 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 <a href="#">ExceptionDef</a> object.
<code>members</code>	A sequence of <a href="#">StructMember</a> structures that describes the members of the new <a href="#">ExceptionDef</a> object.

## Exceptions

<code>BAD_PARAM,</code> minor code 2	An object with the specified <code>id</code> already exists in the repository.
<code>BAD_PARAM,</code> minor code 3	The specified <code>name</code> already exists within this <code>Container</code> and multiple versions are not supported.
<code>BAD_PARAM,</code> minor code 4	The created object is not allowed by the <code>Container</code> . Certain interfaces derived from <code>Container</code> may restrict the types of definitions that they may contain.

## See Also

[CORBA::ExceptionDef](#)

## Container::create\_interface()

```
// IDL
InterfaceDef create_interface(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in InterfaceDefSeq base_interfaces
    in boolean is_abstract
);

// Java
org.omg.CORBA.InterfaceDef create_interface(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    org.omg.CORBA.InterfaceDef[] base_interfaces
```

---

```
);
```

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 repository ID of the new <a href="#">InterfaceDef</a> object. It is an error to specify an ID that already exists within the object's repository.
<code>name</code>	The name of the new <a href="#">InterfaceDef</a> object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.
<code>version</code>	A version for the new <a href="#">InterfaceDef</a> object.
<code>base_interfaces</code>	A sequence of <a href="#">InterfaceDef</a> objects from which the new interface inherits.
<code>is_abstract</code>	If true the interface is abstract.

## Exceptions

<code>BAD_PARAM,</code> minor code 2	An object with the specified <code>id</code> already exists in the repository.
<code>BAD_PARAM,</code> minor code 3	The specified <code>name</code> already exists within this Container and multiple versions are not supported.
<code>BAD_PARAM,</code> minor code 4	The created object is not allowed by the Container. Certain interfaces derived from Container may restrict the types of definitions that they may contain.

## See Also

[CORBA::InterfaceDef](#)

## Container::create\_module()

```
// IDL
ModuleDef create_module (
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version
);
```

---

```
// Java
org.omg.CORBA.ModuleDef create_module(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version
);
```

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. It is an error to specify an ID that already exists within the object's repository.

**name** The name of the new [ModuleDef](#) object. It is an error to 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.

### Exceptions

**BAD\_PARAM,** An object with the specified `id` already exists in the repository.   
minor code 2

**BAD\_PARAM,** The specified `name` already exists within this Container and   
minor code 3 multiple versions are not supported.

**BAD\_PARAM,** The created object is not allowed by the Container. Certain   
minor code 4 interfaces derived from Container may restrict the types of definitions that they may contain.

### Container::create\_native()

```
// IDL
NativeDef create_native(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
);
```

---

```
// Java
org.omg.CORBA.NativeDef create_native(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version
);
```

Creates a [NativeDef](#) 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 [NativeDef](#) object is defined.

## Parameters

**id**            The repository ID of the new [NativeDef](#) object. It is an error to specify an ID that already exists within the object's repository.

**name**            The name of the new [NativeDef](#) object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.

**version**        A version for the [NativeDef](#) object to be created.

## Exceptions

**BAD\_PARAM,**        An object with the specified `id` already exists in the repository.   
minor code 2

**BAD\_PARAM,**        The specified `name` already exists within this Container and   
minor code 3 multiple versions are not supported.

**BAD\_PARAM,**        The created object is not allowed by the Container. Certain   
minor code 4 interfaces derived from Container may restrict the types of definitions that they may contain.

## Container::create\_struct()

```
// IDL
StructDef create_struct(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in StructMemberSeq members
);
```

---

```
// Java
org.omg.CORBA.StructDef create_struct(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    org.omg.CORBA.StructMember[] members
);
```

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.

### Parameters

**id**            The repository ID of the new [StructDef](#) object. It is an error to specify an ID that already exists within the object's repository.

**name**            The name of the new [StructDef](#) object. It is an error to 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 describes the members of the new [StructDef](#) object.

### Exceptions

**BAD\_PARAM,**        An object with the specified `id` already exists in the repository.   
minor code 2

**BAD\_PARAM,**        The specified `name` already exists within this Container and   
minor code 3 multiple versions are not supported.

**BAD\_PARAM,**        The created object is not allowed by the Container. Certain   
minor code 4 interfaces derived from Container may restrict the types of definitions that they may contain.

### See Also

[CORBA::StructDef](#)

### Container::create\_union()

```
// IDL
UnionDef create_union(
    in RepositoryId id,
```

---

```

        in Identifier name,
        in VersionSpec version,
        in IDLType discriminator_type,
        in UnionMemberSeq members
    );

// Java
org.omg.CORBA.UnionDef create_union(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    org.omg.CORBA.IDLType discriminator_type,
    org.omg.CORBA.UnionMember[] members
);

```

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.

## Parameters

<code>id</code>	The repository ID of the new <a href="#">UnionDef</a> object. It is an error to specify an ID that already exists within the object's repository.
<code>name</code>	The name of the new <a href="#">UnionDef</a> object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.
<code>version</code>	A version for the new <a href="#">UnionDef</a> object.
<code>discriminator_type</code>	The type of the union discriminator.
<code>members</code>	A sequence of <a href="#">UnionMember</a> structures that describes the members of the new <a href="#">UnionDef</a> object.

## Exceptions

<code>BAD_PARAM</code> ,	An object with the specified <code>id</code> already exists in the repository.
--------------------------	--

- 
- `BAD_PARAM`,      The specified name already exists within this `Container` and  
minor code 3 multiple versions are not supported.
- `BAD_PARAM`,      The created object is not allowed by the `Container`. Certain  
minor code 4 interfaces derived from `Container` may restrict the types of  
definitions that they may contain.

**See Also**      [CORBA::UnionDef](#)

### **Container::create\_value()**

```
// IDL
ValueDef create_value(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in boolean is_custom,
    in boolean is_abstract,
    in ValueDef base_value,
    in boolean is_truncatable,
    in ValueDefSeq abstract_base_values,
    in InterfaceDef supported_interfaces,
    in InitializerSeq initializers
);

// Java
org.omg.CORBA.ValueDef create_value(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    boolean is_custom,
    boolean is_abstract,
    byte flags,
    org.omg.CORBA.ValueDef base_value,
    boolean has_safe_base,
    org.omg.CORBA.ValueDef[] abstract_base_values,
    org.omg.CORBA.InterfaceDef supported_interfaces,
    org.omg.CORBA.Initializer[] initializers
);
```

---

Creates a new empty [ValueDef](#) 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 [ValueDef](#) object is defined.

## Parameters

<code>id</code>	The repository ID of the new <a href="#">ValueDef</a> object. It is an error to specify an ID that already exists within the object's repository.
<code>name</code>	The name of the new <a href="#">ValueDef</a> object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.
<code>version</code>	A version for the new <a href="#">ValueDef</a> object.
<code>is_custom</code>	If true the value type is custom.
<code>is_abstract</code>	If true the value type is abstract.
<code>base_value</code>	The base value for this value type.
<code>is_truncatable</code>	if true the value type is truncatable.
<code>abstract_base_values</code>	A sequence of <a href="#">ValueDef</a> structures that describes the base values of the new <a href="#">ValueDef</a> object.
<code>supported_interfaces</code>	The interface the value type supports.
<code>initializers</code>	A sequence of initializers for the new <a href="#">ValueDef</a> object.

## Exceptions

<code>BAD_PARAM,</code> minor code 2	An object with the specified <code>id</code> already exists in the repository.
<code>BAD_PARAM,</code> minor code 3	The specified <code>name</code> already exists within this Container and multiple versions are not supported.
<code>BAD_PARAM,</code> minor code 4	The created object is not allowed by the Container. Certain interfaces derived from Container may restrict the types of definitions that they may contain.



---

## Container::create\_value\_box()

```
// IDL
ValueBoxDef create_value_box(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType original_type_def
);

// Java
org.omg.CORBA.ValueBoxDef create_value_box(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    org.omg.CORBA.IDLType original_type_def
);
```

Creates a new empty [ValueBoxDef](#) 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 [ValueBoxDef](#) object is defined.

### Parameters

<code>id</code>	The repository ID of the new <a href="#">ValueBoxDef</a> object. It is an error to specify an ID that already exists within the object's repository.
<code>name</code>	The name of the new <a href="#">ValueBoxDef</a> object. It is an error to 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 <a href="#">ValueBoxDef</a> object.
<code>original_type_def</code>	The IDL data type of the value box.

### Exceptions

`BAD_PARAM`, An object with the specified `id` already exists in the repository.  
minor code 2

- 
- `BAD_PARAM,`      The specified `name` already exists within this `Container` and  
minor code 3 multiple versions are not supported.
- `BAD_PARAM,`      The created object is not allowed by the `Container`. Certain  
minor code 4 interfaces derived from `Container` may restrict the types of  
definitions that they may contain.

## Container::describe\_contents()

```
// IDL
DescriptionSeq describe_contents(
    in DefinitionKind limit_type,
    in boolean exclude_inherited,
    in long max_returned_objs
);

// Java
org.omg.CORBA.ContainerPackage.Description[] describe_contents(
    org.omg.CORBA.DefinitionKind limit_type,
    boolean exclude_inherited,
    int max_returned_objs
);
```

Returns a sequence of structures of type `Container::Description`.  
`describe_contents()` is a combination of operations [Contained::describe\(\)](#)  
and [Container::contents\(\)](#).

### Parameters

<code>limit_type</code>	If this is set to <code>dk_all</code> , then all of the contained interface repository objects are returned. If set to the <a href="#">DefinitionKind</a> for a particular interface repository kind, it returns only objects of that kind. For example, if set to <code>dk_Operation</code> , then it returns contained operations only.
<code>exclude_inherited</code>	Applies only to interfaces. If true, no inherited objects are returned. If false, objects are returned even if they are inherited.
<code>max_returned_objs</code>	The number of objects that can be returned in the call. Setting a value of <code>-1</code> means return all contained objects.

---

**See Also**

[CORBA::Container::contents\(\)](#)  
[CORBA::Contained::describe\(\)](#)

## Container::lookup()

```
// IDL
Contained lookup(
    in ScopedName search_name
);

// Java
org.omg.CORBA.Contained lookup(
    java.lang.String search_name
);
```

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](#).

**See Also**

[CORBA::Container::lookup\\_name\(\)](#)

## Container::lookup\_name()

```
// IDL
ContainedSeq lookup_name (
    in Identifier search_name,
    in long levels_to_search,
    in DefinitionKind limit_type,
    in boolean exclude_inherited
);

// Java
org.omg.CORBA.Contained[] lookup_name(
    java.lang.String search_name,
    int levels_to_search,
```

---

```
    org.omg.CORBA.DefinitionKind limit_type,  
    boolean exclude_inherited  
);
```

Locates an object or objects by name within the target container and returns a sequence of contained objects. The named objects can be directly or indirectly defined in or inherited into the target container. (More than one object, having the same simple name can exist within a nested scope structure.)

## Parameters

<code>search_name</code>	The simple name of the object to search for.
<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 <code>-1</code> , the current object and all contained interface repository objects are searched. If set to <code>1</code> , only the current object is searched.
<code>limit_type</code>	If this is set to <code>dk_all</code> , then all of the contained interface repository objects are returned. If set to the <a href="#">DefinitionKind</a> for a particular interface repository kind, it returns only objects of that kind. For example, if set to <code>dk_Operation</code> , then it returns contained operations only.
<code>exclude_inherited</code>	Applies only to interfaces. If true, no inherited objects are returned. If false, objects are returned even if they are inherited.

**See Also**     [CORBA::DefinitionKind](#)

# CORBA::Context Class

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.

A client can optionally maintain one or more CORBA `Context` objects, that 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 method that takes a `Context` parameter.

You can arrange `Context` objects 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 method can decide whether to use just the context actually passed, or the hierarchy above it.

The `Context` class is as follows:

```
// IDL
pseudo interface Context {
    readonly attribute Identifier context_name;
    readonly attribute Context parent;
    Context create_child(in Identifier child_ctx_name);
    void set_one_value(in Identifier propname, in any propvalue);
    void set_values(in NVList values);
    void delete_values(in Identifier propname);
    NVList get_values(in Identifier start_scope,
        in Flags op_flags,
        in Identifier pattern);
};

// Java
package org.omg.CORBA;
public abstract class Context {
    public abstract String context\_name();
}
```

---

```

public abstract Context parent();
public abstract Context create\_child(
    String child_ctx_name
);
public abstract void set\_one\_value(
    String propname,
    Any propvalue
);
public abstract void set\_values(
    NVList values
);
public abstract void delete\_values(
    String propname
);
public abstract NVList get\_values(
    String start_scpe,
    int op_flags,
    String pattern
);
}

```

### Context::context\_name()

```

// Java
abstract public java.lang.String context_name();

```

Returns the name of the `Context` object. Ownership of the returned value is maintained by the `Context` and must not be freed by the caller.

#### See Also

[CORBA::Context::create\\_child\(\)](#)

### Context::create\_child()

```

// Java
abstract public org.omg.CORBA.Context create_child(
    java.lang.String child_ctx_name
);

```

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\(\)](#) look in parent contexts if necessary to find matching property names.

**Parameters**

`child_ctx_name` The newly created context.

**See Also**

[CORBA::Context::get\\_values\(\)](#)

**Context::delete\_values()**

```
// Java
abstract public void delete_values(
    java.lang.String propname
);
```

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**

`propname` The property name to be deleted. If `prop_name` has a trailing asterisk (\*), all matching properties are deleted.

**Exceptions**

An exception is raised if no matching property is found.

**Context::get\_values()**

```
// Java
abstract public org.omg.CORBA.NVList get_values(
    java.lang.String start_scope,
    int op_flags,
    java.lang.String pattern
);
```

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 in, 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 the value <code>CORBA::CTX_RESTRICT_SCOPE</code> is specified, then searching is limited to the specified search scope or context object.
<code>values</code>	An <a href="#">NVList</a> to contain the returned property values.

## Context::parent()

```
// Java
abstract public org.omg.CORBA.Context parent();
```

Returns the parent of the `Context` object. Ownership of the return value is maintained by the `Context` and must not be freed by the caller.

## See Also

[CORBA::Context::create\\_child\(\)](#)

## Context::set\_one\_value()

```
// Java
abstract public void set_one_value(
    java.lang.String propname,
    org.omg.CORBA.Any propvalue
);
```

Adds a property name and value to the `Context`. Although the value member is of type `Any`, the type of the `Any` must be a string.

## Parameters

<code>propname</code>	The name of the property to add.
<code>propvalue</code>	The value of the property to add.

## See Also

[CORBA::Context::set\\_values\(\)](#)



---

## Context::set\_values()

```
// Java
abstract public void set_values(
    org.omg.CORBA.NVList values
);
```

Sets one or more property values in the `Context`. The previous value of a 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](#).

### See Also

[CORBA::Context::set\\_one\\_value\(\)](#)



# CORBA::ContextList Class

A `ContextList` allows an application to provide a list of [Context](#) strings that must be supplied when a dynamic invocation [Request](#) is invoked.

The [Context](#) is where the actual values are obtained by the ORB. The `ContextList` supplies only the context strings whose values are to be looked up and sent with the request invocation. The serverless `ContextList` object allows the application to specify context information in a way that avoids potentially expensive interface repository lookups for the information by the ORB during a request.

```
// IDL
pseudo interface ContextList {
    readonly attribute unsigned long count;
    void add(in string ctx);
    string item(in unsigned long index) raises (CORBA::Bounds);
    void remove(in unsigned long index) raises (CORBA::Bounds);
};

c
```

## See Also

[CORBA::Object::\\_create\\_request\(\)](#)  
[CORBA::Request::contexts](#)  
[CORBA::ORB::create\\_context\\_list\(\)](#)

## ContextList::add()

```
// Java
abstract public void add(
    java.lang.String ctx
);
```

Adds a context string to the context list.

## Parameters

`ctx`                    A string representing context information.

### ContextList::count()

```
// Java
abstract public int count();
```

Returns the number of context strings in the context list.

### ContextList::item()

```
// Java
abstract public java.lang.String item(
    int index
) throws org.omg.CORBA.Bounds;
```

Returns the context item at the indexed location of the list. This return value must not be released by the caller because ownership of the return value is maintained by the `ContextList`.

#### Parameters

`index`            The indexed location of the desired context item.

### ContextList::remove()

```
// Java
abstract public void remove(
    int index
) throws org.omg.CORBA.Bounds;
```

Removes from the context list the context item at the indexed location.

# CORBA::Current Interface

The `Current` interface is the base interface for providing information about the current thread of execution. Each ORB or CORBA service that needs its own context derives an interface from `Current` to provide information that is associated with the thread of execution in which the ORB or CORBA service is running. Interfaces that derives from `Current` include:

`PortableServer::Current`

Your application can obtain an instance of the appropriate `Current` interface by invoking [resolve\\_initial\\_references\(\)](#).

Operations on interfaces derived from `Current` access the state associated with the thread in which they are invoked, not the state associated with the thread from which the `Current` was obtained.

The IDL interface follows:

```
//IDL
module CORBA {
// interface for the Current object
    interface Current {
        };
    ...
};

// Java
package org.omg.CORBA;
public interface Current extends org.omg.CORBA.Object {}
```

## See Also

`PortableServer::Current`

[CORBA::ORB::resolve\\_initial\\_references\(\)](#)



# CORBA::CustomMarshal Value Type

Custom value types can override the default marshaling/unmarshaling mechanism and provide their own way to encode/decode their state. If an application's value type is marked as custom, you use custom marshaling to facilitate integration of such mechanisms as existing class libraries and other legacy systems. Custom marshaling is not to be used as the standard marshaling mechanism.

`CustomMarshal` is an abstract value type that is meant to be implemented by the application programmer and used by the ORB. For example, if an application's value type needs to use custom marshaling, the IDL declares it explicitly as follows:

```
// Application-specific IDL
custom valuetype type {
    // optional state definition
    ...
};
```

When implementing a custom value type such as this, you must provide a concrete implementation of the `CustomMarshal` operations so that the ORB is able to marshal and unmarshal the value type. Each custom marshaled value type needs its own implementation.

You can use the skeletons generated by the IDL compiler as the basis for your implementation. These operations provide the streams for marshaling. Your implemented `CustomMarshal` code encapsulates the application code that can marshal and unmarshal instances of the value type over a stream using the CDR encoding. It is the responsibility of your implementation to marshal the value type's state of all of its base types (if it has any).

The implementation requirements of the streaming mechanism require that the implementations must be local because local memory addresses such as those for the marshal buffers have to be manipulated by the ORB.

Semantically, `CustomMarshal` is treated as a custom value type's implicit base class, although the custom value type does not actually inherit it in IDL. While nothing prevents you from writing IDL that inherits from

---

`CustomMarshal`, doing so will not in itself make the type custom, nor will it cause the ORB to treat it as a custom value type. You must implement these `CustomMarshal` operations.

Implement the following IDL operations for a custom value type:

```
// IDL in module CORBA
abstract valuetype CustomMarshal {
    void marshal(
        in DataOutputStream os
    );
    void unmarshal(
        in DataInputStream is
    );
};
```

### **CustomMarshal::marshal()**

```
void marshal(org.omg.CORBA.DataOutputStream os);
```

The operation you implement so that the ORB can marshal a custom value type.

#### **Parameters**

`os`                    A handle to the output stream the ORB uses to marshal the custom value type.

Use the operations of the [DataOutputStream](#) in your implementation to write the custom value type's data to the stream as appropriate.

#### **See Also**

[CORBA::DataOutputStream](#)

### **CustomMarshal::unmarshal()**

```
void unmarshal(org.omg.CORBA.DataInputStream is);
```

The operation you implement so that the ORB can unmarshal a custom value type.

#### **Parameters**

`is`                    A handle to the input stream the ORB uses to unmarshal the custom value type.



---

Use the operations of the [DataInputStream](#) in your implementation to read the custom value type's data from the stream as appropriate.

**See Also**     [CORBA](#) : [DataInputStream](#)



# CORBA::DataInputStream Value Type

The `DataInputStream` value type is a stream used by [unmarshal\(\)](#) for unmarshaling an application's custom value type. You use the `DataInputStream` operations in your implementation of [unmarshal\(\)](#) to read specific types of data from the stream, as defined in the custom value type. The stream takes care of breaking the data into chunks if necessary. The IDL code is as follows:

```
// IDL in module CORBA
abstract valuetype DataInputStream {
    any read\_any\(\);
    boolean read\_boolean\(\);
    char read\_char\(\);
    wchar read\_wchar\(\);
    octet read\_octet\(\);
    short read\_short\(\);
    unsigned short read\_ushort\(\);
    long read\_long\(\);
    unsigned long read\_ulong\(\);
    unsigned long long read\_ulonglong\(\);
    float read\_float\(\);
    double read\_double\(\);
    long double read\_longdouble\(\);
    string read\_string\(\);
    wstring read\_wstring\(\);
    Object read\_Object\(\);
    AbstractBase read\_Abstract\(\);
    ValueBase read\_Value\(\);
    TypeCode read\_TypeCode\(\);

    void read\_any\_array(
        inout AnySeq seq,
        in unsigned long offset,
        in unsigned long length
    );
    void read\_boolean\_array(
        inout BooleanSeq seq,
        in unsigned long offset,
```

---

```
        in unsigned long length
    );
void read\_char\_array(
    inout CharSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read\_wchar\_array(
    inout WcharSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read\_octet\_array(
    inout OctetSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read\_short\_array(
    inout ShortSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read\_ushort\_array(
    inout UShortSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read\_long\_array(
    inout LongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read\_ulong\_array(
    inout ULongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read\_ulonglong\_array(
    inout ULongLongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read\_longlong\_array(
```

---

```
        inout LongLongSeq seq,
        in unsigned long offset,
        in unsigned long length
    );
void read_float_array(
    inout FloatSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read_double_array(
    inout DoubleSeq seq,
    in unsigned long offset,
    in unsigned long length
);
};
```

### Exceptions

MARSHAL            An inconsistency is detected for any operations.

### See Also

[CORBA::CustomMarshal](#)  
[CORBA::DataOutputStream](#)

### DataInputStream::read\_any()

```
// IDL
any read_any();

// Java
org.omg.CORBA.Any read_any();
```

Returns an any data type from the stream.

### DataInputStream::read\_any\_array()

```
// IDL
void read_any_array(
    inout AnySeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

---

```
// Java
void read_any_array(
    org.omg.CORBA.AnySeqHolder seq,
    int offset,
    int length );
```

Reads an array of `any` data from the stream.

### Parameters

<code>seq</code>	The sequence into which the data is placed.
<code>offset</code>	The starting index from which to read from the sequence.
<code>length</code>	The number of items to read from the array.

### **DataInputStream::read\_boolean()**

```
// IDL
boolean read_boolean();

// Java
boolean read_boolean();
```

Returns a `boolean` data type from the stream.

### **DataInputStream::read\_boolean\_array()**

```
// IDL
void read_boolean_array(
    inout BooleanSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void read_boolean_array(
    org.omg.CORBA.BooleanSeqHolder seq,
    int offset,
    int length );
```

Reads an array of `boolean` data from the stream.

---

## Parameters

<code>seq</code>	The sequence into which the data is placed.
<code>offset</code>	The starting index from which to read from the sequence.
<code>length</code>	The number of items to read from the array.

## **DataInputStream::read\_char()**

```
// IDL
char read_char();

// Java
char read_char();
```

Returns a `char` data type from the stream.

## **DataInputStream::read\_char\_array()**

```
// IDL
void read_char_array(
    inout CharSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void read_char_array(
    org.omg.CORBA.CharSeqHolder seq,
    int offset,
    int length );
```

Reads an array of `char` data from the stream.

## Parameters

<code>seq</code>	The sequence into which the data is placed.
<code>offset</code>	The starting index from which to read from the sequence.
<code>length</code>	The number of items to read from the array.

---

## **DataInputStream::read\_double()**

```
// IDL
double read_double();

// Java
double read_double();
```

Returns a `double` data type from the stream.

## **DataInputStream::read\_double\_array()**

```
// IDL
void read_double_array(
    inout DoubleSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void read_double_array(
    org.omg.CORBA.DoubleSeqHolder seq,
    int offset,
    int length );
```

Reads an array of `double` data from the stream.

### **Parameters**

<code>seq</code>	The sequence into which the data is placed.
<code>offset</code>	The starting index from which to read from the sequence.
<code>length</code>	The number of items to read from the array.

## **DataInputStream::read\_float()**

```
// IDL
float read_float();

// Java
float read_float();
```

Returns a `float` data type from the stream.



---

## DataInputStream::read\_float\_array()

```
// IDL
void read_float_array(
    inout FloatSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void read_float_array(
    org.omg.CORBA.FloatSeqHolder seq,
    int offset,
    int length );
```

Reads an array of `float` data from the stream.

### Parameters

<code>seq</code>	The sequence into which the data is placed.
<code>offset</code>	The starting index from which to read from the sequence.
<code>length</code>	The number of items to read from the array.

## DataInputStream::read\_long()

```
// IDL
long read_long();

// Java
int read_long();
```

Returns a `long` data type from the stream.

## DataInputStream::read\_long\_array()

```
// IDL
void read_long_array(
    inout LongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

---

```
// Java
void read_long_array(
    org.omg.CORBA.LongSeqHolder seq,
    int offset,
    int length );
```

Reads an array of `long` data from the stream.

### Parameters

<code>seq</code>	The sequence into which the data is placed.
<code>offset</code>	The starting index from which to read from the sequence.
<code>length</code>	The number of items to read from the array.

### **DataInputStream::read\_longdouble()**

```
// IDL
long double read_longdouble();
```

```
// Java
```

Unsupported.

### **DataInputStream::read\_longlong\_array()**

```
// IDL
void read_longlong_array(
    inout LongLongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

```
// Java
void read_longlong_array(
    org.omg.CORBA.LongLongSeqHolder seq,
    int offset,
    int length );
```

Reads an array of `long long` data from the stream.

---

## Parameters

<code>seq</code>	The sequence into which the data is placed.
<code>offset</code>	The starting index from which to read from the sequence.
<code>length</code>	The number of items to read from the array.

### **DataInputStream::read\_Object()**

```
// IDL
Object read_Object();

// Java
org.omg.CORBA.Object read_objref();
```

Returns an [Object](#) (object reference) data type from the stream.

### **DataInputStream::read\_octet()**

```
// IDL
octet read_octet();

// Java
byte read_octet();
```

Returns an octet data type from the stream.

### **DataInputStream::read\_octet\_array()**

```
// IDL
void read_octet_array(
    inout OctetSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void read_octet_array(
    org.omg.CORBA.OctetSeqHolder seq,
    int offset,
```

---

```
int length );
```

Reads an array of `octet` data from the stream.

### Parameters

<code>seq</code>	The sequence into which the data is placed.
<code>offset</code>	The starting index from which to read from the sequence.
<code>length</code>	The number of items to read from the array.

### **DataInputStream::read\_short()**

```
// IDL
short read_short();
```

```
// Java
short read_short();
```

Returns a `short` data type from the stream.

### **DataInputStream::read\_short\_array()**

```
// IDL
void read_short_array(
    inout ShortSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

```
// Java
void read_short_array(
    org.omg.CORBA.ShortSeqHolder seq,
    int offset,
    int length );
```

Reads an array of `short` data from the stream.

### Parameters

<code>seq</code>	The sequence into which the data is placed.
------------------	---

---

offset	The starting index from which to read from the sequence.
length	The number of items to read from the array.

### **DataInputStream::read\_string()**

```
// IDL
string read_string();

// Java
java.lang.String read_string();
```

Returns a `string` data type from the stream.

### **DataInputStream::read\_TypeCode()**

```
// IDL
TypeCode read_TypeCode();

// Java
org.omg.CORBA.TypeCode read_TypeCode();
```

Returns a [TypeCode](#) data type from the stream.

### **DataInputStream::read\_ulong()**

```
// IDL
unsigned long read_ulong();

// Java
int read_ulong();
```

Returns an `unsigned long` data type from the stream.

### **DataInputStream::read\_ulong\_array()**

```
// IDL
void read_ulong_array(
    inout ULongSeq seq,
    in unsigned long offset,
```

---

```
        in unsigned long length
    );

    // Java
    void read_ulong_array(
        org.omg.CORBA.ULongSeqHolder seq,
        int offset,
        int length );
```

Reads an array of unsigned long data from the stream.

### Parameters

seq	The sequence into which the data is placed.
offset	The starting index from which to read from the sequence.
length	The number of items to read from the array.

### **DataInputStream::read\_ulonglong()**

```
// IDL
unsigned long long read_ulonglong();

// Java
long read_ulonglong();
```

Returns an unsigned long long data type from the stream.

### **DataInputStream::read\_ulonglong\_array()**

```
// IDL
void read_ulonglong_array(
    inout ULongLongSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void read_ulonglong_array(
    org.omg.CORBA.ULongLongSeqHolder seq,
    int offset,
    int length );
```

---

Reads an array of unsigned long long data from the stream.

### Parameters

seq	The sequence into which the data is placed.
offset	The starting index from which to read from the sequence.
length	The number of items to read from the array.

### DataInputStream::read\_ushort()

```
// IDL
unsigned short read_ushort();

// Java
short read_ushort();
```

Returns an unsigned short data type from the stream.

### DataInputStream::read\_ushort\_array()

```
// IDL
void read_ushort_array(
    inout UShortSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void read_ushort_array(
    org.omg.CORBA.UShortSeqHolder seq,
    int offset,
    int length );
```

Reads an array of unsigned short data from the stream.

### Parameters

seq	The sequence into which the data is placed.
offset	The starting index from which to read from the sequence.
length	The number of items to read from the array.

---

## DataInputStream::read\_Value()

```
// IDL
ValueBase read_Value();

// Java
java.io.Serializable read_value();
```

Returns a value type from the stream.

## DataInputStream::read\_wchar()

```
// IDL
wchar read_wchar();

// Java
char read_wchar();
```

Returns a `wchar` data type from the stream.

## DataInputStream::read\_wchar\_array()

```
// IDL
void read_wchar_array(
    inout WCharSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void read_wchar_array(
    org.omg.CORBA.WCharSeqHolder seq,
    int offset,
    int length );
```

Reads an array of `wchar` data from the stream.

### Parameters

<code>seq</code>	The sequence into which the data is placed.
<code>offset</code>	The starting index from which to read from the sequence.
<code>length</code>	The number of items to read from the array.



---

## **DataInputStream::read\_wstring()**

```
// IDL
wstring read_wstring();

// Java
java.lang.String read_wstring();
```

Returns a `wstring` data type from the stream.



# CORBA::DataOutputStream Value Type

The `DataOutputStream` value type is a stream used by [marshal\(\)](#) for marshaling an application's custom value type. You use the `DataOutputStream` operations in your implementation of [marshal\(\)](#) to write specific types of data to the stream, as defined in the custom value type. The stream takes care of breaking the data into chunks if necessary. The IDL code is as follows:

```
//IDL in module CORBA
abstract valuetype DataOutputStream {
    void write\_any( in any value );
    void write\_boolean( in boolean value );
    void write\_char( in char value );
    void write\_wchar( in wchar value );
    void write\_octet( in octet value );
    void write\_short( in short value );
    void write\_ushort( in unsigned short value );
    void write\_long( in long value );
    void write\_ulong( in unsigned long value );
    void write\_longlong( in long long value );
    void write\_ulonglong( in unsigned long long value );
    void write\_float( in float value );
    void write\_double( in double value );
    void write\_string( in string value );
    void write\_wstring( in wstring value );
    void write\_Object( in Object value );
    void write\_Value( in ValueBase value );
    void write\_TypeCode( in TypeCode value );
    void write\_any\_array(
        in AnySeq seq,
        in unsigned long offset,
        in unsigned long length );
    void write\_boolean\_array(
        in BooleanSeq seq,
        in unsigned long offset,
```

---

```
        in unsigned long length );
void write\_char\_array(
    in CharSeq seq,
    in unsigned long offset,
    in unsigned long length );
void write\_wchar\_array(
    in WcharSeq seq,
    in unsigned long offset,
    in unsigned long length );
void write\_octet\_array(
    in OctetSeq seq,
    in unsigned long offset,
    in unsigned long length );
void write\_short\_array(
    in ShortSeq seq,
    in unsigned long offset,
    in unsigned long length );
void write\_ushort\_array(
    in UShortSeq seq,
    in unsigned long offset,
    in unsigned long length );
void write\_long\_array(
    in LongSeq seq,
    in unsigned long offset,
    in unsigned long length );
void write\_ulong\_array(
    in ULongSeq seq,
    in unsigned long offset,
    in unsigned long length );
void write\_ulonglong\_array(
    in ULongLongSeq seq,
    in unsigned long offset,
    in unsigned long length );
void write\_longlong\_array(
    in LongLongSeq seq,
    in unsigned long offset,
    in unsigned long length );
void write\_float\_array(
    in FloatSeq seq,
    in unsigned long offset,
    in unsigned long length );
void write\_double\_array(
    in DoubleSeq seq,
```

---

```
        in unsigned long offset,  
        in unsigned long length );  
};
```

### Exceptions

MARSHAL            An inconsistency is detected for any operations.

### See Also

[CORBA::CustomMarshal](#)  
[CORBA::DataInputStream](#)

## DataOutputStream::write\_any()

```
// IDL  
void write_any(  
    in any value  
);  
  
// Java  
void write_any( org.omg.CORBA.Any val );
```

Writes an *any* data type to the stream.

### Parameters

value            The value written to the stream.

## DataOutputStream::write\_any\_array()

```
// IDL  
void write_any_array(  
    in AnySeq seq,  
    in unsigned long offset,  
    in unsigned long length  
);  
  
// Java  
void write_any_array( org.omg.CORBA.Any[] buf,  
    int offset, int len );
```

Writes an array of *any* data to the stream.

---

## Parameters

<code>seq</code>	The sequence of data to write to the stream.
<code>offset</code>	The offset in <code>seq</code> from which to start writing data.
<code>length</code>	The number of data items to write.

## **DataOutputStream::write\_boolean()**

```
// IDL
void write_boolean(
    in boolean value
);

// Java
void write_boolean( boolean val );
```

Writes a `boolean` data type to the stream.

## Parameters

<code>value</code>	The value written to the stream.
--------------------	----------------------------------

## **DataOutputStream::write\_boolean\_array()**

```
// IDL
void write_boolean_array(
    in BooleanSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void write_boolean_array( boolean[] buf, int offset, int len );
```

Writes an array of `boolean` data to the stream.

## Parameters

<code>seq</code>	The sequence of data to write to the stream.
<code>offset</code>	The offset in <code>seq</code> from which to start writing data.
<code>length</code>	The number of data items to write.

---

## **DataOutputStream::write\_char()**

```
// IDL
void write_char(
    in char value
);

// Java
void write_char( char val );
```

Writes a `char` data type to the stream.

### **Parameters**

`value`            The value written to the stream.

## **DataOutputStream::write\_char\_array()**

```
// IDL
void write_char_array(
    in CharSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void write_char_array( char[] buf, int offset, int len );
```

Writes an array of `char` data to the stream.

### **Parameters**

`seq`                The sequence of data to write to the stream.  
`offset`            The offset in `seq` from which to start writing data.  
`length`            The number of data items to write.

## **DataOutputStream::write\_double()**

```
// IDL
void write_double(
    in double value
);
```

---

```
// Java
void write_double( double val );
```

Writes a `double` data type to the stream.

### Parameters

value            The value written to the stream.

### **DataOutputStream::write\_double\_array()**

```
// IDL
void write_double_array(
    in DoubleSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void write_double_array( double[] buf, int offset, int len );
```

Writes an array of `double` data to the stream.

### Parameters

seq              The sequence of data to write to the stream.  
offset            The offset in `seq` from which to start writing data.  
length            The number of data items to write.

### **DataOutputStream::write\_float()**

```
// IDL
void write_float(
    in float value
);

// Java
void write_float( float val );
```

Writes a `float` data type to the stream.



---

## Parameters

value            The value written to the stream.

## DataOutputStream::write\_float\_array()

```
// IDL
void write_float_array(
    in FloatSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void write_float_array( float[] buf, int offset, int len );
```

Writes an array of `float` data to the stream.

## Parameters

seq            The sequence of data to write to the stream.  
offset        The offset in `seq` from which to start writing data.  
length        The number of data items to write.

## DataOutputStream::write\_long()

```
// IDL
void write_long(
    in long value
);

// Java
void write_long( int val );
```

Writes a `long` data type to the stream.

## Parameters

value            The value written to the stream.

---

## DataOutputStream::write\_long\_array()

```
// IDL
void write_long_array(
    in LongSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void write_long_array( int[] buf, int offset, int len );
```

Writes an array of long data to the stream.

### Parameters

seq	The sequence of data to write to the stream.
offset	The offset in seq from which to start writing data.
length	The number of data items to write.

## DataOutputStream::write\_longlong()

```
// IDL
void write_longlong(
    in long long value
);

// Java
void write_longlong( long val );
```

Writes a long long data type to the stream.

### Parameters

value	The value written to the stream.
-------	----------------------------------

## DataOutputStream::write\_longlong\_array()

```
// IDL
void write_longlong_array(
    in LongLongSeq seq,
    in unsigned long offset,
```

---

```
        in unsigned long length
    );
// Java
void write_longlong_array( long[] buf, int offset, int len );
```

Writes an array of `long long` data to the stream.

### Parameters

<code>seq</code>	The sequence of data to write to the stream.
<code>offset</code>	The offset in <code>seq</code> from which to start writing data.
<code>length</code>	The number of data items to write.

### **DataOutputStream::write\_Object()**

```
// IDL
void write_Object(
    in Object value
);
// Java
void write_objref( org.omg.CORBA.Object val );
```

Writes an [Object](#) data type (object reference) to the stream.

### Parameters

<code>value</code>	The value written to the stream.
--------------------	----------------------------------

### **DataOutputStream::write\_octet()**

```
// IDL
void write_octet(
    in octet value
);
// Java
void write_octet( byte val );
```

Writes an `octet` data type to the stream.

---

## Parameters

value            The value written to the stream.

### **DataOutputStream::write\_octet\_array()**

```
// IDL
void write_octet_array(
    in OctetSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void write_octet_array( byte[] buf, int offset, int len );
```

Writes an array of octet data to the stream.

## Parameters

seq            The sequence of data to write to the stream.  
offset        The offset in seq from which to start writing data.  
length        The number of data items to write.

### **DataOutputStream::write\_short()**

```
// IDL
void write_short(
    in short value
);

// Java
void write_short( short val );
```

Writes a short data type to the stream.

## Parameters

value            The value written to the stream.

---

## DataOutputStream::write\_short\_array()

```
// IDL
void write_short_array(
    in ShortSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void write_short_array( short[] buf, int offset, int len );
```

Writes an array of `short` data to the stream.

### Parameters

<code>seq</code>	The sequence of data to write to the stream.
<code>offset</code>	The offset in <code>seq</code> from which to start writing data.
<code>length</code>	The number of data items to write.

## DataOutputStream::write\_string()

```
// IDL
void write_string(
    in string value
);

// Java
void write_string( java.lang.String val );
```

Writes a `string` data type to the stream.

### Parameters

<code>value</code>	The value written to the stream.
--------------------	----------------------------------

## DataOutputStream::write\_TypeCode()

```
// IDL
void write_TypeCode(
    in TypeCode value
);
```

---

```
// Java
void write_TypeCode( org.omg.CORBA.TypeCode val );
```

Writes a [TypeCode](#) data type to the stream.

### Parameters

value            The value written to the stream.

## DataOutputStream::write\_ulong()

```
// IDL
void write_ulong(
    in unsigned long value
);
```

```
// Java
void write_ulong( int val );
```

Writes an `unsigned long` data type to the stream.

### Parameters

value            The value written to the stream.

## DataOutputStream::write\_ulong\_array()

```
// IDL
void write_ulong_array(
    in ULongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

```
// Java
void write_ulong_array( int[] buf, int offset, int len );
```

Writes an array of `unsigned long` data to the stream.

### Parameters

seq              The sequence of data to write to the stream.

---

offset	The offset in <code>seq</code> from which to start writing data.
length	The number of data items to write.

### **DataOutputStream::write\_ulonglong()**

```
// IDL
void write_ulonglong(
    in unsigned long long value
);

// Java
void write_ulonglong( long val );
```

Writes an unsigned long long data type to the stream.

#### **Parameters**

value	The value written to the stream.
-------	----------------------------------

### **DataOutputStream::write\_ulonglong\_array()**

```
// IDL
void write_ulonglong_array(
    in ULongLongSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void write_ulonglong_array( long[] buf, int offset, int len );
```

Writes an array of unsigned long long data to the stream.

#### **Parameters**

seq	The sequence of data to write to the stream.
offset	The offset in <code>seq</code> from which to start writing data.
length	The number of data items to write.

---

## DataOutputStream::write\_ushort()

```
// IDL
void write_ushort(
    in unsigned short value
);

// Java
void write_ushort( short val );
```

Writes an `unsigned short` data type to the stream.

### Parameters

`value`            The value written to the stream.

## DataOutputStream::write\_ushort\_array()

```
// IDL
void write_ushort_array(
    in UShortSeq seq,
    in unsigned long offset,
    in unsigned long length
);

// Java
void write_ushort_array( short[] buf, int offset, int len );
```

Writes an array of `unsigned short` data to the stream.

### Parameters

`seq`                The sequence of data to write to the stream.  
`offset`            The offset in `seq` from which to start writing data.  
`length`            The number of data items to write.

## DataOutputStream::write\_Value()

```
// IDL
void write_Value(
    in ValueBase value
);
```



---

```
// Java
void write_value( java.io.Serializable vb );
```

Writes a value type to the stream.

### Parameters

value            The value written to the stream.

### **DataOutputStream::write\_wchar()**

```
// IDL
void write_wchar(
    in wchar value
);
```

```
// Java
void write_wchar( char val );
```

Writes a `wchar` data type to the stream.

### Parameters

value            The value written to the stream.

### **DataOutputStream::write\_wchar\_array()**

```
// IDL
void write_wchar_array(
    in WCharSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

```
// Java
void write_wchar_array( char[] buf, int offset, int len );
```

Writes an array of `wchar` data to the stream.

### Parameters

seq              The sequence of data to write to the stream.

---

offset	The offset in <code>seq</code> from which to start writing data.
length	The number of data items to write.

### **DataOutputStream::write\_wstring()**

```
// IDL
void write_wstring(
    in wstring value
);

// Java
void write_wstring( java.lang.String val );
```

Writes a `wstring` data type to the stream.

#### **Parameters**

value	The value written to the stream.
-------	----------------------------------

# CORBA::DomainManager Interface

The `DomainManager` interface provides an operation to find the [Policy](#) objects associated with a policy domain. Each policy domain includes one policy domain manager object (`DomainManager`). The `DomainManager` has associated with it the policy objects for that domain and it records the membership of the domain.

```
// IDL in CORBA Module
interface DomainManager {
    Policy get\_domain\_policy(
        in PolicyType policy_type
    );
};
```

A *policy domain* is a set of objects with an associated set of policies. These objects are the *policy domain members*. The policies represent the rules and criteria that constrain activities of the objects of the policy domain. Policy domains provide a higher granularity for policy management than an individual object instance provides.

When a new object reference is created, the ORB implicitly associates the object reference (and hence the object that it is associated with) with one or more policy domains, thus defining all the policies to which the object is subject. If an object is simultaneously a member of more than one policy domain, it is governed by all policies of all of its domains.

The `DomainManager` does not include operations to manage domain membership, structure of domains, or to manage which policies are associated with domains. However, because a `DomainManager` is a CORBA object, it has access to the `CORBA::Object` interface, which is available to all CORBA objects. The [Object](#) interface includes the following related operations:

[\\_get\\_domain\\_managers\(\)](#) allows your applications to retrieve the domain managers and hence the security and other policies applicable to individual objects that are members of the policy domain.

You can also obtain an object's policy using [\\_get\\_policy\(\)](#).

---

## DomainManager::get\_domain\_policy()

```
Policy get_domain_policy (  
    in PolicyType policy_type  
);  
  
// Java  
org.omg.CORBA.Policy get_domain_policy(  
    int policy_type  
);
```

Returns a reference to the policy object of the specified policy type for objects in this policy domain.

### Parameters

`policy_type`     The type of policy for objects in the domain which the application wants to administer.

There may be several policies associated with a domain, with a policy object for each. There is at most one policy of each type associated with a policy domain. The policy objects are thus shared between objects in the domain, rather than being associated with individual objects. Consequently, if an object needs to have an individual policy, then it must be a singleton member of a policy domain.

### Exceptions

`INV_POLICY`     The value of policy type is not valid either because the specified type is not supported by this ORB or because a policy object of that type is not associated with this object.

### See Also

[CORBA::Policy](#)  
[CORBA::Object::get\\_domain\\_managers\(\)](#)  
[CORBA::Object::get\\_policy\(\)](#)

# CORBA::EnumDef Interface

Interface `EnumDef` describes an IDL enumeration definition in the interface repository.

```
// IDL in module CORBA.  
interface EnumDef : TypedefDef {  
    attribute EnumMemberSeq members;  
};
```

The inherited operation [describe\(\)](#) is also described.

## EnumDef::describe()

```
// IDL  
Description describe();
```

Inherited from [Contained](#) (which [TypedefDef](#) inherits), `describe()` returns a `Description`. The [DefinitionKind](#) for the description's `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](#). The `type` field of the struct gives the [TypeCode](#) of the defined enumeration.

## See Also

[CORBA::TypedefDef::describe\(\)](#)

## EnumDef::members Attribute

```
// IDL  
attribute EnumMemberSeq members;  
  
// Java  
java.lang.String[] members();  
void members(java.lang.String[] _val);
```

Returns or changes the enumeration's list of identifiers (its set of enumerated constants).



# CORBA::Environment Class

The `Environment` class provides a way to handle exceptions in situations where true exception-handling mechanisms are unavailable or undesirable.

For example, in the DII exceptions raised by remote invocation are stored in an `Environment` member variable in the `Request` object after the invocation returns. DII clients should test the value of this `Environment` variable by calling the `env()` method on the `Request` object. If the returned `java.lang.Exception` is null, no exception was raised. If it is not null, the returned exception should be examined and acted on in an appropriate manner.

```
// IDL
pseudo interface Environment {
    attribute exception exception;
    void clear();
};
// Java
package org.omg.CORBA;
abstract public class Environment {
    abstract public void clear();
    public abstract void exception(
        java.lang.Exception except );
    public abstract java.lang.Exception exception();
}
```

**See Also** [CORBA::ORB::create\\_environment\(\)](#)

## Environment::clear()

```
//Java
abstract public void clear();
```

Deletes the Exception, if any, contained in the `Environment`. This is equivalent to passing zero to [exception\(\)](#). It is not an error to call [clear\(\)](#) on an `Environment` that holds no exception.

**See Also** [CORBA::Environment::exception\(\)](#)

---

## Environment::exception()

```
// Java  
public abstract java.lang.Exception exception();
```

Extracts the exception contained in the `Environment` object.

```
//Java  
public abstract void exception( java.lang.Exception except );
```

Sets the exception member variable in the `Environment` object to `except`.

### Parameters

<code>except</code>	The <code>Exception</code> assigned to the <code>Environment</code> . The <code>Environment</code> does not copy the parameter but it assumes ownership of it. The <code>Exception</code> must be dynamically allocated.
---------------------	--

### See Also

[CORBA::Environment::clear\(\)](#)



# CORBA::Exception Class

Details of this class can be found in the CORBA specification. The C++ Language Mapping document provides the following explanation of the CORBA::Exception class:

```
// C++
class Exception
{
    public:
    virtual ~Exception();
    virtual void _raise() const = 0;
    virtual const char * _name() const;
    virtual const char * _rep_id() const;
};
```

The `Exception` base class is abstract and may not be instantiated except as part of an instance of a derived class. It supplies one pure virtual function to the exception hierarchy: the `_raise()` function. This function can be used to tell an exception instance to throw itself so that a catch clause can catch it by a more derived type.

Each class derived from `Exception` implements `_raise()` as follows:

```
// C++
void SomeDerivedException::_raise() const
{
    throw *this;
}
```

For environments that do not support exception handling, please refer to Section 1.42.2, "Without Exception Handling," on page 1-169 of the CORBA specification for information about the `_raise()` function.

The `_name()` function returns the unqualified (unscoped) name of the exception. The `_rep_id()` function returns the repository ID of the exception.



# CORBA::ExceptionDef Interface

Interface `ExceptionDef` describes an IDL exception in the interface repository. It inherits from interface [Contained](#) and [Container](#).

```
// IDL in module CORBA.  
interface ExceptionDef : Contained, Container {  
    readonly attribute TypeCode type;  
    attribute StructMemberSeq members;  
};
```

The inherited operation [describe\(\)](#) is also described.

## See Also

[CORBA::Contained](#)  
[CORBA::Container](#)

## ExceptionDef::describe()

```
// IDL  
Description describe();
```

Inherited from [Contained](#), `describe()` returns a `Description`.

The [DefinitionKind](#) for the `kind` member of this structure is `dk_Exception`. The `value` member is an `any` whose [TypeCode](#) is `_tc_ExceptionDescription` and whose value is a structure of type [ExceptionDescription](#).

The `type` field of the [ExceptionDescription](#) structure gives the [TypeCode](#) of the defined exception.

## See Also

[CORBA::Contained::describe\(\)](#)  
[CORBA::TypeCode](#)

## ExceptionDef::members Attribute

```
// IDL  
attribute StructMemberSeq members;
```

---

```
// Java
org.omg.CORBA.StructMember[] members();
void members(org.omg.CORBA.StructMember[] _val);
```

In a sequence of [StructMember](#) structures, the `members` attribute describes the exception's members.

The `members` attribute can be modified to change the structure's members.

**See Also**

[CORBA::StructDef](#)  
[CORBA::ExceptionDef::type](#)

## ExceptionDef::type Attribute

```
// IDL
readonly attribute TypeCode type;

// Java
org.omg.CORBA.TypeCode type();
```

The type of the exception (from which the definition of the exception can be understood). The [TypeCode](#) kind for an exception is `tk_except`.

**See Also**

[CORBA::TypeCode](#)  
[CORBA::ExceptionDef::members](#)

# CORBA::ExceptionList Class

An `ExceptionList` object allows an application to provide a list of `TypeCodes` for all application-specific (user-defined) exceptions that may result when a dynamic invocation [Request](#) is invoked. This server-less `ExceptionList` object allows the ORB to avoid potentially expensive interface repository lookups for the exception information during a request.

```
// PIDL
pseudo interface ExceptionList {
    readonly attribute unsigned long count;
    void add(in TypeCode exc);
    TypeCode item(in unsigned long index) raises(Bounds);
    void remove(in unsigned long index) raises(Bounds);
};
```

## See Also

[CORBA::Object::\\_create\\_request\(\)](#)  
[CORBA::Request::exceptions](#)  
[CORBA::ORB::create\\_exception\\_list\(\)](#)

## ExceptionList::add()

```
// Java
abstract public void add(org.omg.CORBA.TypeCode exc);
```

Adds a [TypeCode](#) to the exception list.

## Parameters

`exc`            The `TypeCode` to be added to the list. Should be a `TypeCode` for an exception.

## ExceptionList::count()

```
// Java
abstract public int count();
```

Returns the number of items in the exception list.

---

## ExceptionList::item()

```
// Java
abstract public org.omg.CORBA.TypeCode item(int index)
    throws org.omg.CORBA.Bounds;
```

Returns the exception item at the indexed location of the list. This return value must not be released by the caller because ownership of the return value is maintained by the `ExceptionList`.

### Parameters

`index`            The indexed location of the desired item.

## ExceptionList::remove()

```
// Java
abstract public void remove(int index)
    throws org.omg.CORBA.Bounds;
```

Removes from the exception list the item at the indexed location.

### Parameters

`index`            The indexed location of the desired item.

# CORBA::FixedDef Interface

The FixedDef interface describes an IDL fixed-point type in the interface repository. A fixed-point decimal literal consists of an integer part, a decimal point, a fraction part, and a d or D.

```
// IDL in module CORBA.  
interface FixedDef : IDLType {  
    attribute unsigned short digits;  
    attribute short scale;  
};
```

The inherited [IDLType](#) attribute is a tk\_fixed [TypeCode](#), which describes a fixed-point decimal number.

## See Also

[CORBA::Repository::create\\_fixed\(\)](#)

## FixedDef::digits Attribute

```
// IDL  
attribute unsigned short digits;  
  
// Java  
short digits();  
void digits( short _val );
```

The `digits` attribute specifies the total number of decimal digits in the fixed-point number, and must be in the range of 1 to 31, inclusive.

## FixedDef::scale Attribute

```
// IDL  
attribute short scale;  
  
// Java  
short scale();  
void scale( short _val );
```

The `scale` attribute specifies the position of the decimal point.





# CORBA.InterfaceDefPackage.FullInterfaceDescription Class

## InterfaceDefPackage.FullInterfaceDescription.FullInterfaceDescription()

```
// IDL
struct FullInterfaceDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    OpDescriptionSeq operations;
    AttrDescriptionSeq attributes;
    RepositoryIdSeq base_interfaces;
    TypeCode type;
    boolean is_abstract;
};
// Java
public FullInterfaceDescription(
    java.lang.String name,
    java.lang.String id,
    java.lang.String defined_in,
    java.lang.String version,
    org.omg.CORBA.OperationDescription[] operations,
    org.omg.CORBA.AttributeDescription[] attributes,
    java.lang.String[] base_interfaces,
    org.omg.CORBA.TypeCode type,
    boolean is_abstract
)
```

Describes an interface including its operations and attributes.

name	The name of the interface.
id	An identifier of the interface.

---

<code>defined_in</code>	The identifier where the interface is defined.
<code>version</code>	The version of the interface.
<code>operations</code>	A sequence of interface operations.
<code>attributes</code>	A sequence of interface attributes.
<code>base_interfaces</code>	A sequence of base interfaces from which this interface is derived.
<code>type</code>	The type of the interface.
<code>is_abstract</code>	True if the interface is an abstract one, false otherwise.

**See Also** [CORBA::InterfaceDef::describe\\_interface\(\)](#)

# CORBA::IDLType Interface

The abstract base interface `IDLType` describes interface repository objects that represent IDL types. These types include interfaces, type definitions, structures, unions, enumerations, and others. Thus, the `IDLType` is a base interface for the following interfaces:

- [ArrayDef](#)
- [AliasDef](#)
- [EnumDef](#)
- [FixedDef](#)
- [InterfaceDef](#)
- [NativeDef](#)
- [PrimitiveDef](#)
- [SequenceDef](#)
- [StringDef](#)
- [StructDef](#)
- [TypedefDef](#)
- [UnionDef](#)
- [ValueBoxDef](#)
- [ValueDef](#)
- [WstringDef](#)

The `IDLType` provides access to the [TypeCode](#) describing the type, and is used in defining other interfaces wherever definitions of IDL types must be referenced.

```
// IDL in module CORBA.  
interface IDLType : IRObjct {  
    readonly attribute TypeCode type;  
};
```

## See Also

- [CORBA::IRObjct](#)
- [CORBA::TypeCode](#)
- [CORBA::TypedefDef](#)

---

## IDLType::type Attribute

```
//IDL  
readonly attribute TypeCode type;  
  
// Java  
org.omg.CORBA.TypeCode type();
```

Encodes the type information of an interface repository object. Most type information can also be extracted using operations and attributes defined for derived types of the [IDLType](#).

### See Also

[CORBA::TypeCode](#)

# CORBA::InterfaceDef Interface

InterfaceDef describes an IDL interface definition in the interface repository. It may contain lists of constants, typedefs, exceptions, operations, and attributes. It inherits from the interfaces [Container](#), [Contained](#), and [IDLType](#).

Calling [\\_get\\_interface\(\)](#) on a reference to an object returns a reference to the InterfaceDef object that defines the CORBA object's interface.

```
// IDL in module CORBA.
interface InterfaceDef : Container, Contained, IDLType {
    // read/write interface
    attribute InterfaceDefSeq base\_interfaces;

    // read interface
    boolean is\_a(
        in RepositoryId interface_id
    );

    struct FullInterfaceDescription {
        Identifier name;
        RepositoryId id;
        RepositoryId defined_in;
        VersionSpec version;
        OpDescriptionSeq operations;
        AttrDescriptionSeq attributes;
        RepositoryIdSeq base_interfaces;
        TypeCode type;
    };

    FullInterfaceDescription describe\_interface();

    // write interface
    AttributeDef create\_attribute(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in IDLType type,
        in AttributeMode mode
    );
};
```

---

```

);

OperationDef create\_operation(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType result,
    in OperationMode mode,
    in ParDescriptionSeq params,
    in ExceptionDefSeq exceptions,
    in ContextIdSeq contexts
);
}; // End interface InterfaceDef

```

The inherited operation [describe\(\)](#) is also described.

**See Also**

[CORBA::Contained](#)  
[CORBA::Container](#)  
[CORBA::Object::\\_get\\_interface\(\)](#)

## InterfaceDef::base\_interfaces Attribute

```

// IDL
attribute InterfaceDefSeq base_interfaces;

// Java
void base_interfaces(org.omg.CORBA.InterfaceDef[] _val);

```

The `base_interfaces` attribute lists in a sequence of `InterfaceDef` objects the interfaces from which this interface inherits.

The inheritance specification of an `InterfaceDef` object can be changed by changing its `base_interfaces` attribute.

**Exceptions**

`BAD_PARAM,`      The name of any definition contained in the interface conflicts  
minor code 5      with the `name` of a definition in any of the base interfaces.

**See Also**

[CORBA::Object::\\_get\\_interface\(\)](#)

---

## InterfaceDef::create\_attribute()

```
// IDL
AttributeDef create_attribute(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType type,
    in AttributeMode mode
);

// Java
org.omg.CORBA.AttributeDef create_attribute(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    org.omg.CORBA.IDLType type,
    org.omg.CORBA.AttributeMode mode
);
```

Creates a new [AttributeDef](#) within the target `InterfaceDef`. The [defined\\_in](#) attribute of the new [AttributeDef](#) is set to the target `InterfaceDef`.

### Parameters

<code>id</code>	The identifier of the new attribute. It is an error to specify an <code>id</code> that already exists within the target object's repository.
<code>name</code>	The name of the attribute. It is an error to specify a <code>name</code> that already exists within this <code>InterfaceDef</code> .
<code>version</code>	A version for this attribute.
<code>type</code>	The <a href="#">IDLType</a> for this attribute.
<code>mode</code>	Specifies whether the attribute is read only ( <a href="#">ATTR_READONLY</a> ) or read/write ( <a href="#">ATTR_NORMAL</a> ).

### Exceptions

<code>BAD_PARAM,</code> minor code 2	An object with the specified <code>id</code> already exists in the repository.
<code>BAD_PARAM,</code> minor code 3	An object with the same <code>name</code> already exists in this <code>InterfaceDef</code> .

**See Also** [CORBA::AttributeDef](#)

---

## InterfaceDef::create\_operation()

```
// IDL
OperationDef create_operation(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType result,
    in OperationMode mode,
    in ParDescriptionSeq params,
    in ExceptionDefSeq exceptions,
    in ContextIdSeq contexts
);

// Java
org.omg.CORBA.OperationDef create_operation(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    org.omg.CORBA.IDLType result,
    org.omg.CORBA.OperationMode mode,
    org.omg.CORBA.ParameterDescription[] params,
    org.omg.CORBA.ExceptionDef[] exceptions,
    java.lang.String[] contexts
);
```

Creates a new [OperationDef](#) within the target `InterfaceDef`. The [defined\\_in](#) attribute of the new [OperationDef](#) is set to the target `InterfaceDef`.

### Parameters

<code>id</code>	The identifier of the new attribute. It is an error to specify an <code>id</code> that already exists within the target object's repository.
<code>name</code>	The name of the attribute. It is an error to specify a <code>name</code> that already exists within this <code>InterfaceDef</code> .
<code>version</code>	A version number for this operation.
<code>result</code>	The return type for this operation.
<code>mode</code>	Specifies whether this operation is normal ( <a href="#">OP_NORMAL</a> ) or oneway ( <a href="#">OP_ONEWAY</a> ).
<code>params</code>	A sequence of <a href="#">ParameterDescription</a> structures that describes the parameters to this operation.



---

exceptions A sequence of [ExceptionDef](#) objects that describes the exceptions this operation can raise.

contexts A sequence of context identifiers for this operation.

**See Also**

[CORBA::OperationDef](#)  
[CORBA::ExceptionDef](#)

## InterfaceDef::describe()

```
// IDL
Description describe();
```

Inherited from [Contained](#), `describe()` returns a `Description`. The `DefinitionKind` for the `kind` member is `dk_Interface`. The value member is an any whose `TypeCode` is `_tc_InterfaceDescription` and whose value is a structure of type [InterfaceDescription](#).

**See Also**

[CORBA::Contained::describe\(\)](#)

## InterfaceDef::describe\_interface()

```
// IDL
FullInterfaceDescription describe_interface();

// Java
org.omg.CORBA.InterfaceDefPackage.FullInterfaceDescription
    describe_interface();
```

Returns a description of the interface, including its operations, attributes, and base interfaces in a [FullInterfaceDescription](#).

Details of exceptions and contexts can be determined via the returned sequence of [OperationDescription](#) structures.

**See Also**

[CORBA::OperationDef::describe\(\)](#)  
[CORBA::AttributeDef::describe\(\)](#)

## InterfaceDef::FullInterfaceDescription

See the [“CORBA.InterfaceDefPackage.FullInterfaceDescription Class”](#).

---

## InterfaceDef::is\_a()

```
// IDL
boolean is_a(
    in RepositoryId interface_id
);

// Java
boolean is_a(java.lang.String interface_id);
```

Returns `TRUE` if the interface is either identical to or inherits (directly or indirectly) from the interface represented by `interface_id`. Otherwise the operation returns `FALSE`.

### Parameters

`interface_id` The repository ID of another `InterfaceDef` object.

# CORBA::IObject Interface

The interface `IObject` is the base interface from which all interface repository interfaces are derived.

```
// IDL in module CORBA.  
  
interface IObject {  
    readonly attribute DefinitionKind def\_kind;  
    void destroy();  
};
```

## IObject::def\_kind Attribute

```
// IDL  
readonly attribute DefinitionKind def_kind;  
  
// Java  
org.omg.CORBA.DefinitionKind def_kind();
```

Identifies the kind of an IFR object. For example, an [OperationDef](#) object, describing an IDL operation, has the kind `dk_Operation`.

### See Also

[CORBA::DefinitionKind](#)

## IObject::destroy()

```
// IDL  
void destroy();  
  
// Java  
void destroy();
```

Deletes an IFR object. This also deletes any objects contained within the target object.

### Exceptions

`BAD_INV_ORDER` with a minor value of:

2                    `destroy()` is invoked on a [Repository](#) or on a [PrimitiveDef](#) object.

---

1

An attempt is made to destroy an object that would leave the repository in an incoherent state.

# CORBA::ModuleDef Interface

The interface `ModuleDef` describes an IDL module in the interface repository. It inherits from the interfaces [Container](#) and [Contained](#).

```
// IDL in module CORBA.  
interface ModuleDef : Container, Contained { };
```

The inherited operation [describe\(\)](#) is also described.

## ModuleDef::describe()

```
// IDL  
Description describe();
```

Inherited from [Contained](#), `describe()` returns a `Description`.

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](#).

### See Also

[CORBA::Contained::describe\(\)](#)



# CORBA::NamedValue Class

A `NamedValue` object describes an argument to a request or a return value, especially in the DII, and is used as an element of an [NVList](#) object. A `NamedValue` object maintains an any value, parameter-passing mode flags, and an (optional) name.

```
// IDL
pseudo interface NamedValue {
    readonly attribute Identifier name;
    readonly attribute any value;
    readonly attribute Flags flags;
};
```

## See Also

[CORBA::NVList](#)  
[CORBA::ORB::create\\_named\\_value\(\)](#)  
[CORBA::Request::result\(\)](#)  
[CORBA::Object::\\_create\\_request\(\)](#)

## NamedValue::flags()

```
// Java
abstract public int flags();
```

Returns the flags associated with the `NamedValue`. Flags identify the parameter passing mode for arguments of an [NVList](#).

## NamedValue::name()

```
// Java
abstract public java.lang.String name();
```

Returns the (optional) name associated with the `NamedValue`. This is the name of a parameter or argument to a request.

---

## NamedValue::value()

```
// Java  
abstract public org.omg.CORBA.Any value();
```

Returns a reference to the `org.omg.CORBA.Any` object contained in the `NamedValue`.



# CORBA::NativeDef Interface

The interface `NativeDef` describes an IDL native type in the interface repository. It inherits from the interface [TypedefDef](#). The inherited type attribute is a `tk_native` [TypeCode](#) that describes the native type.

```
// IDL in module CORBA
interface NativeDef : TypedefDef {};
```

**See Also** [CORBA::Container::create\\_native\(\)](#)



# CORBA::NVList Class

An `NVList` is a pseudo-object used for constructing parameter lists. It is a list of [NamedValue](#) elements where each [NamedValue](#) describes an argument to a request.

The [NamedValue](#) and `NVList` types are used mostly in the DII in the request operations to describe arguments and return values. They are also used in the context object routines to pass lists of property names and values. The `NVList` is also used in the DSI operation [ServerRequest::arguments\(\)](#).

The `NVList` class is partially opaque and may only be created by using [ORB::create\\_list\(\)](#). The `NVList` class is as follows:

```
// IDL
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 item(in unsigned long index) raises(Bounds);
    void remove(in unsigned long index) raises(Bounds);
};
```

## See Also

[CORBA::NamedValue](#)  
[CORBA::ORB::create\\_list\(\)](#)  
[CORBA::Object::\\_\\_create\\_request\(\)](#)

## **NVList::count()**

```
abstract public int count();
```

Returns the number of elements in the list.

---

## NVList::add()

```
// Java
abstract public org.omg.CORBA.NamedValue add( int flgs );
```

Creates an unnamed value, initializes only the flags, and adds it to the list. The new [NamedValue](#) is returned.

### Parameters

flgs

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.

### See Also

[CORBA::NVList::add\\_item\(\)](#)  
[CORBA::NVList::add\\_value\(\)](#)

## NVList::add\_item()

```
// Java
abstract public org.omg.CORBA.NamedValue add_item(
    java.lang.String item_name, int flgs );
```

Creates and returns a [NamedValue](#) with name and flags initialized, and adds it to the list.

### Parameters

item\_name      Name of item.

flgs

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.

### See Also

[CORBA::NVList::add\(\)](#)  
[CORBA::NVList::add\\_value\(\)](#)

## NVList::add\_value()

```
// Java
abstract public org.omg.CORBA.NamedValue add_value(
```

---

```
java.lang.String item_name,  
org.omg.CORBA.Any val, int flgs );
```

Creates and returns a [NamedValue](#) with name, value, and flags initialized and adds it to the list.

### Parameters

`item_name`      Name of item.  
`value`            Value of item.  
`flags`

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.

### See Also

[CORBA::NVList::add\(\)](#)  
[CORBA::NVList::add\\_item\(\)](#)

## NVList::item()

```
// Java  
abstract public org.omg.CORBA.NamedValue item( int index )  
    throws org.omg.CORBA.Bounds;
```

Returns the [NamedValue](#) list item at the given index. The first item is at index 0. This method can be used to access existing elements in the list.

### Parameters

`index`            Index of item.

## NVList::remove()

```
// Java  
abstract public void remove( int index )  
    throws org.omg.CORBA.Bounds;
```

Removes the item at the given index. The first item is at index 0.

---

**Parameters**

`index`

Index of item

# CORBA::Object Class

The `Object` class is the base class for all normal CORBA objects. This class has some common methods that operate on any CORBA object. These operations are implemented directly by the ORB, not passed on to your object's implementation.

On the client side, the methods of this class are called on a proxy (unless collocation is set). On the server side, they are called on the real object.

Table 4 shows the methods provided by the `CORBA::Object` class:

**Table 4:** *Methods of the Object Class*

Manage Object References	Create Requests for the DII
<a href="#"><u>duplicate()</u></a>	<a href="#"><u>create_request()</u></a>
<a href="#"><u>hash()</u></a>	<a href="#"><u>request()</u></a>
<a href="#"><u>is_a()</u></a>	
<a href="#"><u>is_equivalent()</u></a>	<b>Access Information in the IFR</b>
<a href="#"><u>non_existent()</u></a>	
<a href="#"><u>release()</u></a>	<a href="#"><u>get_interface()</u></a>
<b>Manage Policies and Domains</b>	
<a href="#"><u>get_domain_managers()</u></a>	
<a href="#"><u>get_policy()</u></a>	

```
// IDL
interface Object {
    boolean is_nil();
    Object duplicate();
    void release();
    ImplementationDef get_implementation();
    InterfaceDef get_interface();
    boolean is_a(in string logical_type_id);
    boolean non_existent();
}
```

---

```

boolean is_equivalent(in Object other_object);
unsigned long hash(in unsigned long maximum);
void create_request(
    in Context ctx,
    in Identifier operation,
    in NVList arg_list,
    in NamedValue result,
    out Request request,
    in Flags req_flags
);
void create_request2(
    in Context ctx,
    in Identifier operation,
    in NVList arg_list,
    in NamedValue result,
    in ExceptionList exclist,
    in ContextList ctxtlist,
    out Request request,
    in Flags req_flags
);
Policy_ptr get_policy(in PolicyType policy_type);
DomainManagerList get_domain_managers();
Object set_policy_overrides(
    in PolicyList policies,
    in SetOverrideType set_or_add
);

// IDL Additions from CORBA Messaging
Policy get_policy(
    in PolicyType type
);
Policy get_client_policy(
    in PolicyType type
);
Object set_policy_overrides(
    in PolicyList policies,
    in SetOverrideType set_add
)
    raises (InvalidPolicies);
PolicyList get_policy_overrides(
    in PolicyTypeSeq types
);
boolean validate_connection(

```



---

```

        out PolicyList inconsistent_policies
    );
};
// Java
package org.omg.CORBA;
public interface Object {
    boolean _is_a(String Identifier);
    boolean _is_equivalent(Object that);
    boolean _non_existent();
    int _hash(int maximum);
    org.omg.CORBA.Object _duplicate();
    void _release();
    org.omg.CORBA.Object _get_interface_def();
    Request _request(String s);
    Request _create_request(Context ctx,
        String operation,
        NVList arg_list,
        NamedValue result);
    Request _create_request(Context ctx,
        String operation,
        NVList arg_list,
        NamedValue result,
        ExceptionList exclist,
        ContextList ctxlist);
    Policy _get_policy(int policy_type);
    DomainManager[] _get_domain_managers();
    org.omg.CORBA.Object _set_policy_override(
        Policy[] policies,
        SetOverrideType set_add);
}

```

### **Object::\_create\_request()**

```

// Java
Request _create_request(Context ctx,
    String operation,
    NVList arg_list,
    NamedValue result);

Request _create_request(Context ctx,
    String operation,
    NVList arg_list,

```

---

```
NamedValue result,  
ExceptionList exclist,  
ContextList ctxlist);
```

These construct a CORBA: [Request](#) object. These methods are part of the DII and create an ORB request on an object by constructing one of the object's operations.

See [\\_request\(\)](#) for a simpler alternative way to create a [Request](#).

The only implicit object reference operations allowed with the `_create_request()` call include:

[\\_non\\_existent\(\)](#)  
[\\_is\\_a\(\)](#)  
[\\_get\\_interface\(\)](#)

## Exceptions

**BAD\_PARAM** The name of an implicit operation that is not allowed is passed to [\\_create\\_request\(\)](#)—for example, `_is_equivalent` is passed to [\\_create\\_request\(\)](#) as the `operation` parameter.

## See Also

[CORBA::Object::\\_request\(\)](#)  
[CORBA::Request](#)  
[CORBA::Request::arguments\(\)](#)  
[CORBA::Request::ctx\(\)](#)  
[CORBA::NVList](#)  
[CORBA::NamedValue](#)

## Object::\_duplicate()

```
// Java  
org.omg.CORBA.Object _duplicate();
```

Returns a new reference to `obj` and increments the reference count of the object. Because object references are opaque and ORB-dependent, it is not possible for your application to allocate storage for them. Therefore, if more than one copy of an object reference is needed, use this method to create a duplicate.

---

## Object::\_get\_domain\_managers()

```
DomainManager[] _get_domain_managers();
```

Returns the list of immediately enclosing domain managers of this object. At least one domain manager is always returned in the list since by default each object is associated with at least one domain manager at creation.

The `_get_domain_managers()` method allows applications such as administration services to retrieve the domain managers and hence the security and other policies applicable to individual objects that are members of the domain.

**See Also** [CORBA::DomainManager](#)

## Object::\_get\_interface()

```
// Java
InterfaceDef _get_interface();
```

Returns a reference to an object in the interface repository that describes this object's interface.

**See Also** [CORBA::InterfaceDef](#)

## Object::\_get\_policy()

```
// Java
Policy _get_policy(int policy_type);
```

Returns a reference to the [Policy](#) object of the type specified by the `policy_type` parameter.

### Parameters

`policy_type`    The type of policy to get.

`_get_policy()` returns the effective policy which is the one that would be used if a request were made. Note that the effective policy may change from invocation to invocation due to transparent rebinding. Invoking [\\_non\\_existent\(\)](#) on an object reference prior to `_get_policy()` ensures the accuracy of the returned effective policy.

---

Quality of Service (see “[Quality of Service Framework](#)”) is managed on a per-object reference basis with `_get_policy()`.

## Exceptions

`INV_POLICY` The value of `policy_type` is not valid either because the specified type is not supported by this ORB or because a policy object of that type is not associated with this object.

**See Also** [CORBA::Object::\\_non\\_existent\(\)](#)

## Object::\_hash()

```
// Java
int _hash(int maximum);
```

Returns a hashed value for the object reference in the range 0...maximum.

## Parameters

`maximum` The maximum value that is to be returned from the hash method.

Use `_hash()` to quickly guarantee that objects references refer to different objects. For example, if `_hash()` returns the same hash number for two object references, the objects might or might not be the same, however, if the method returns different numbers for object references, these object references are guaranteed to be for different objects.

In order to efficiently manage large numbers of object references, some applications need to support a notion of object reference identity. Object references are associated with internal identifiers that you can access indirectly by using `_hash()`. The value of this internal identifier does not change during the lifetime of the object reference.

You can use `_hash()` and [\\_is\\_equivalent\(\)](#) to support efficient maintenance and search of tables keyed by object references. `_hash()` allows you to partition the space of object references into sub-spaces of potentially equivalent object references. For example, setting `maximum` to 7 partitions the object reference space into a maximum of 8 sub-spaces (0 - 7).

**See Also** [CORBA::Object::\\_is\\_equivalent\(\)](#)

---

## Object::\_is\_a()

```
// Java
boolean _is_a(String Identifier);
```

Returns 1 (true) if the target object is either an instance of the type specified in `logical_type_id` or of a derived type of the type in `logical_type_id`. If the target object is neither, it returns 0 (false).

### Parameters

`Identifier`      The fully scoped name of the IDL interface. This is a string denoting a shared type identifier (`RepositoryId`). Use an underscore ('\_') rather than a scope operator (::) to delimit the scope.

The ORB maintains type-safety for object references over the scope of an ORB, but you can use this method to help maintaining type-safety when working in environments that do not have compile time type checking to explicitly maintain type safety.

### Exceptions

If `_is_a()` cannot make a reliable determination of type compatibility due to failure, it raises an exception in the calling application code. This enables the application to distinguish among the true, false, and indeterminate cases.

### See Also

[CORBA::Object::\\_non\\_existent\(\)](#)

## Object::\_is\_equivalent()

```
// Java
boolean _is_equivalent(Object that);
```

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. Two objects are equivalent if they have the same object reference, or they both refer to the same object.

### Parameters

`other_object`    An object reference of other object.

---

A typical application use of `_is_equivalent()` is to match object references in a hash table. Bridges could use the method to shorten the lengths of chains of proxy object references. Externalization services could use it to flatten graphs that represent cyclical relationships between objects.

**See Also**

[CORBA::Object::\\_is\\_a\(\)](#)  
[CORBA::Object::\\_hash\(\)](#)

### **Object::\_non\_existent()**

```
// Java
boolean _non_existent();
```

Returns 1 (true) if the object does not exist or returns 0 (false) otherwise.

Normally you might invoke this method on a proxy to determine whether the real object still exists. This method may be used to test whether an object has been destroyed because the method does not raise an exception if the object does not exist.

Applications that maintain state that includes object references, (such as bridges, event channels, and base relationship services) might use this method to sift through object tables for objects that no longer exist, deleting them as they go, as a form of garbage collection.

### **Object::\_release()**

```
// Java
void _release();
```

Signals that the caller is done using this object reference, so internal ORB resources associated with this object reference can be released. Note that the object implementation is not involved in this operation, and other references to the same object are not affected.

---

## Object::\_request()

Request \_request(String operation);

Returns a reference to a constructed [Request](#) on the target object. This is the simpler form of [\\_create\\_request\(\)](#).

### Parameters

operation      The name of the operation.

You can add arguments and contexts after construction using [Request::arguments\(\)](#) and [Request::ctx\(\)](#).

### See Also

[CORBA::Object::\\_create\\_request\(\)](#)

[CORBA::Request::arguments\(\)](#)

[CORBA::Request::ctx\(\)](#)

Returns true if the current effective policies for the object will allow an invocation to be made. Returns false if the current effective policies would cause an invocation to raise the system exception `INV_POLICY`.

### Parameters

inconsistent\_policies      If the current effective policies are incompatible, This parameter contains those policies causing the incompatibility. This returned list of policies is not guaranteed to be exhaustive.

If the object reference is not yet bound, a binding will occur as part of this operation. If the object reference is already bound, but current policy overrides have changed or for any other reason the binding is no longer valid, a rebind will be attempted regardless of the setting of any `RebindPolicy` override. This method is the only way to force such a rebind when implicit rebinds are disallowed by the current effective `RebindPolicy`.

### Exceptions

The appropriate system exception is raised if the binding fails due to some reason unrelated to policy overrides.





# CORBA::OperationDef Interface

Interface `OperationDef` describes an IDL operation that is defined in an IDL interface stored in the interface repository.

One way you can use the `OperationDef` is to construct an [NVList](#) for a specific operation for use in the Dynamic Invocation Interface. For details see [ORB::create\\_operation\\_list\(\)](#).

```
// IDL in module CORBA.
interface OperationDef : Contained {
    readonly attribute TypeCode result;
    attribute IDLType result\_def;
    attribute ParDescriptionSeq params;
    attribute OperationMode mode;
    attribute ContextIdSeq contexts;
    attribute ExceptionDefSeq exceptions;
};
```

The inherited operation [describe\(\)](#) is also described.

## See Also

[CORBA::Contained](#)  
[CORBA::ORB::create\\_operation\\_list\(\)](#)  
[CORBA::ExceptionDef](#)

## OperationDef::contexts Attribute

```
// IDL
attribute ContextIdSeq contexts;

// Java
java.lang.String[] contexts();
void contexts( java.lang.String[] _val );
```

The list of context identifiers specified in the context clause of the operation.

---

## OperationDef::exceptions Attribute

```
// IDL
attribute ExceptionDefSeq exceptions;

// Java
org.omg.CORBA.ExceptionDef[] exceptions();
void exceptions( org.omg.CORBA.ExceptionDef[] _val );
```

The list of exceptions that the operation can raise.

### See Also

[CORBA::ExceptionDef](#)

## OperationDef::describe()

```
// IDL
Description describe();
```

Inherited from [Contained](#), `describe()` returns a `Description`.

The [DefinitionKind](#) for the `kind` member of this structure is `dk_Operation`. The `value` member is an `any` whose [TypeCode](#) is `_tc_OperationDescription` and whose value is a structure of type [OperationDescription](#).

### See Also

[CORBA::Contained::describe\(\)](#)  
[CORBA::ExceptionDef](#)

## OperationDef::mode Attribute

```
// IDL
attribute OperationMode mode;

// Java
org.omg.CORBA.OperationMode mode();
void mode( org.omg.CORBA.OperationMode _val );
```

Specifies whether the operation is normal ([OP\\_NORMAL](#)) or oneway ([OP\\_ONEWAY](#)).

## OperationDef::params Attribute

```
// IDL
attribute ParDescriptionSeq params;
```

---

```
// Java
org.omg.CORBA.ParameterDescription[] params();
void params( org.omg.CORBA.ParameterDescription[] _val );
```

Specifies the parameters for this operation. It is a sequence of structures of type [ParameterDescription](#).

The `name` member of the [ParameterDescription](#) structure 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 `ParameterDescription`s is significant.

**See Also**

[CORBA::TypeCode](#)  
[CORBA::IDLType](#)

### **OperationDef::result Attribute**

```
// IDL
readonly attribute TypeCode result;

// Java
org.omg.CORBA.TypeCode result();
```

The return type of this operation. The attribute `result_def` contains the same information.

**See Also**

[CORBA::TypeCode](#)  
[CORBA::OperationDef::result\\_def](#)

### **OperationDef::result\_def Attribute**

```
// IDL
attribute IDLType result_def;

// Java
org.omg.CORBA.IDLType result_def();
void result_def( org.omg.CORBA.IDLType _val );
```

Describes the return type for this operation. The attribute `result` contains the same information.

Setting the `result_def` attribute also updates the `result` attribute.

---

**See Also**

[CORBA::IDLType](#)

[CORBA::OperationDef::result](#)

# CORBA::ORB Class

The ORB class provides a set of methods and data types that control the ORB from both the client and the server. See [Table 5](#):

**Table 5:** *Methods and Types of the ORB Class*

<b>Object Reference Manipulation</b>	<b>ORB Operation and Threads</b>
<a href="#">_duplicate()</a> <a href="#">list_initial_services()</a> <a href="#">_nil()</a> ObjectId type ObjectIdList sequence <a href="#">object_to_string()</a> <a href="#">resolve_initial_references()</a> <a href="#">string_to_object()</a>	<a href="#">destroy()</a> <a href="#">perform_work()</a> <a href="#">run()</a> <a href="#">shutdown()</a> <a href="#">work_pending()</a>
	<b>ORB Policies and Services</b>
	<a href="#">create_policy()</a>
<b>Dynamic Invocation Interface (DII)</b>	<b>TypeCode Creation Methods</b>
<a href="#">create_environment()</a> <a href="#">create_exception_list()</a> <a href="#">create_list()</a> <a href="#">create_named_value()</a> <a href="#">create_operation_list()</a> <a href="#">get_next_response()</a> <a href="#">poll_next_response()</a> <a href="#">send_multiple_requests_deferred()</a> <a href="#">send_multiple_requests_oneway()</a>	<a href="#">create_abstract_interface_tc()</a> <a href="#">create_alias_tc()</a> <a href="#">create_array_tc()</a> <a href="#">create_enum_tc()</a> <a href="#">create_exception_tc()</a> <a href="#">create_fixed_tc()</a> <a href="#">create_interface_tc()</a> <a href="#">create_native_tc()</a> <a href="#">create_recursive_tc()</a> <a href="#">create_sequence_tc()</a> <a href="#">create_string_tc()</a> <a href="#">create_struct_tc()</a> <a href="#">create_union_tc()</a> <a href="#">create_value_box_tc()</a> <a href="#">create_value_tc()</a> <a href="#">create_wstring_tc()</a>
<b>Value Type Factory Methods</b>	
<a href="#">lookup_value_factory()</a> <a href="#">register_value_factory()</a> <a href="#">unregister_value_factory()</a>	

There are also methods to manage dynamic any data types.

---

You initialize the ORB using [ORB.init\(\)](#).

The ORB class is defined as follows:

```
//IDL
pseudo interface ORB {
    typedef string ObjectId;
    typedef sequence <ObjectId> ObjectIdList;
    exception InconsistentTypeCode {};
    exception InvalidName {};
    string object_to_string ( in Object obj );
    Object string_to_object ( in string str );

    // Dynamic Invocation related operations
    void create\_list ( in long count, out NVList new_list );
    void create\_operation\_list (
        in OperationDef oper,
        out NVList new_list );
    void get\_default\_context ( out Context ctx );
    void send\_multiple\_requests\_oneway(in RequestSeq req);
    void send\_multiple\_requests\_deferred(in RequestSeq req);
    boolean poll\_next\_response();
    void get\_next\_response(out Request req);

    // Service information operations
    boolean get\_service\_information (
        in ServiceType service_type,
        out ServiceInformation service_information );
    ObjectIdList list\_initial\_services ();

    // Initial reference operation
    Object resolve\_initial\_references (
        in ObjectId identifier
        ) raises (InvalidName);

    // Type code creation operations
    TypeCode create\_struct\_tc (
        in RepositoryId id,
        in Identifier name,
        in StructMemberSeq members );
    TypeCode create\_union\_tc (
        in RepositoryId id,
        in Identifier name,
```

---

```
        in TypeCode discriminator_type,
        in UnionMemberSeq members );
TypeCode create\_enum\_tc (
    in RepositoryId id,
    in Identifier name,
    in EnumMemberSeq members );
TypeCode create\_alias\_tc (
    in RepositoryId id,
    in Identifier name,
    in TypeCode original_type );
TypeCode create\_exception\_tc (
    in RepositoryId id,
    in Identifier name,
    in StructMemberSeq members );
TypeCode create\_interface\_tc (
    in RepositoryId id,
    in Identifier name );
TypeCode create\_string\_tc ( in unsigned long bound );
TypeCode create\_wstring\_tc ( in unsigned long bound );
TypeCode create\_fixed\_tc (
    in unsigned short digits,
    in short scale );
TypeCode create\_sequence\_tc (
    in unsigned long bound,
    in TypeCode element_type );
TypeCode create\_recursive\_sequence\_tc ( // deprecated
    in unsigned long bound,
    in unsigned long offset );
TypeCode create\_array\_tc (
    in unsigned long length,
    in TypeCode element_type );
TypeCode create\_value\_tc (
    in RepositoryId id,
    in Identifier name,
    in ValueModifier type_modifier,
    in TypeCode concrete_base,
    in ValueMemberSeq members );
TypeCode create\_value\_box\_tc (
    in RepositoryId id,
    in Identifier name,
    in TypeCode boxed_type );
TypeCode create\_native\_tc (
    in RepositoryId id,
```

---

```

        in Identifier name );
TypeCode create\_recursive\_tc (
    in RepositoryId id );
TypeCode create\_abstract\_interface\_tc (
    in RepositoryId id,
    in Identifier name );

// Thread related operations
boolean work\_pending();
void perform\_work();
void run();
void shutdown( in boolean wait_for_completion );
void destroy();

// Policy related operations
Policy create\_policy(
    in PolicyType type,
    in any val ) raises (PolicyError);

// Dynamic Any related operations deprecated and removed
// from primary list of ORB operations
// Value factory operations
ValueFactory register\_value\_factory(
    in RepositoryId id,
    in ValueFactory factory );
void unregister\_value\_factory(in RepositoryId id);
ValueFactory lookup\_value\_factory(in RepositoryId id);

// Additional operations that only appear in the Java mapping
TypeCode get\_primitive\_tc(in TCKind tcKind);
ExceptionList create\_exception\_list();
ContextList create\_context\_list();
Environment create\_environment();
Current get\_current();
Any create\_any();
OutputStream create\_output\_stream();
void connect(Object obj);
void disconnect(Object obj);
Object get\_value\_def(in String repid);
void (Object wrapper);

// additional methods for ORB initialization go here, but only
// appear in the mapped Java (seeSection 1.21.9, "ORB

```



---

```

// Initialization) Java signatures
// public static ORB init(Strings[] args, Properties props);
// public static ORB init(Applet app, Properties props);
// public static ORB init();
// abstract protected void set_parameters(String[] args,
// java.util.Properties props);
// abstract protected void set_parameters(java.applet.Applet app,
// java.util.Properties props);
};

// Java
package org.omg.CORBA;
public abstract class ORB {
    public abstract org.omg.CORBA.Object
    string_to_object(String str);
    public abstract String
    object_to_string(org.omg.CORBA.Object obj);

    // Dynamic Invocation related operations
    public abstract NVList create_list(int count);

    public NVList create_operation_list(
        org.omg.CORBA.Object oper);

    // oper must really be an OperationDef
    public abstract NamedValue create_named_value(
        String name, Any value, int flags);
    public abstract ExceptionList create_exception_list();
    public abstract ContextList create_context_list();
    public abstract Context get\_default\_context();
    public abstract Environment create_environment();
    public abstract void send_multiple_requests_oneway(
        Request[] req);
    public abstract void send_multiple_requests_deferred(
        Request[] req);
    public abstract boolean poll_next_response();
    public abstract Request get_next_response() throws
        org.omg.CORBA.WrongTransaction;

    // Service information operations
    public boolean get_service_information(
        short service_type,
        ServiceInformationHolder service_info) {

```

---

```

        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public abstract String[] list_initial_services();

    // Initial reference operation
    public abstract org.omg.CORBA.Object
    resolve_initial_references(String object_name)
        throws org.omg.CORBA.ORBPackage.InvalidName;

    // typecode creation
    public abstract TypeCode create_struct_tc(
        String id, String name, StructMember[] members);
    public abstract TypeCode create_union_tc(
        String id,
        String name,
        TypeCode discriminator_type,
        UnionMember[] members);
    public abstract TypeCode create_enum_tc(
        String id,
        String name,
        String[] members);
    public abstract TypeCode create_alias_tc(
        String id,
        String name,
        TypeCode original_type);
    public abstract TypeCode create_exception_tc(
        String id,
        String name,
        StructMember[] members);
    public abstract TypeCode create_interface_tc(
        String id, String name);
    public abstract TypeCode create_string_tc(int bound);
    public abstract TypeCode create_wstring_tc(int bound);
    public TypeCode create_fixed_tc(
        short digits,
        short scale) {
        throw new org.omg.CORBA.NO_IMPLEMENT(); }
    public abstract TypeCode create_sequence_tc(
        int bound, TypeCode element_type);

    public abstract TypeCode create_array_tc(
        int length, TypeCode element_type);
    public TypeCode create_value_tc(

```

---

```

        String id,
        String name,
        short type_modifier,
        TypeCode concrete_base,
        ValueMember[] members) {
        throw new org.omg.CORBA.NO_IMPLEMENT(); }
public TypeCode create_value_box_tc(
    String id,
    String name,
    TypeCode boxed_type) {
    throw new org.omg.CORBA.NO_IMPLEMENT(); }
public TypeCode create_native_tc(
    String id,
    String name) {
    throw new org.omg.CORBA.NO_IMPLEMENT(); }
public TypeCode create_recursive_tc(
    String id) {
    throw new org.omg.CORBA.NO_IMPLEMENT(); }
public TypeCode create_abstract_interface_tc(
    String id,
    String name) {
    throw org.omg.CORBA.NO_IMPLEMENT(); }

// Thread related operations
public boolean work_pending() {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
public void perform_work() {
    throw new org.omg.CORBA.NO_IMPLEMENT(); }
public void run() {
    throw new org.omg.CORBA.NO_IMPLEMENT(); }
public void shutdown(boolean wait_for_completion) {
    throw new org.omg.CORBA.NO_IMPLEMENT(); }
public void destroy() {
    throw new org.omg.CORBA.NO_IMPLEMENT(); }

// Policy related operations
public Policy create_policy(short policy_type, Any val)
    throws org.omg.CORBA.PolicyError {
    throw new org.omg.CORBA.NO_IMPLEMENT(); }

// additional methods for IDL/Java mapping
public abstract TypeCode get_primitive_tc(TCKind tcKind);

```

---

```

public abstract Any create_any();
public abstract org.omg.CORBA.portable.OutputStream
create_output_stream();

// additional static methods for ORB initialization
public static ORB init(
    Strings[] args,
    Properties props);
public static ORB init(
    Applet app,
    Properties props);
public static ORB init();
abstract protected void set_parameters(
    String[] args,
    java.util.Properties props);
abstract protected void set_parameters(
    java.applet.Applet app,
    java.util.Properties props);
}

package org.omg.CORBA_2_3;
public abstract class ORB extends org.omg.CORBA.ORB {
    // always return a ValueDef or throw BAD_PARAM if
    // repid not of a value
    public org.omg.CORBA.Object get_value_def(
        String repid)
        throws org.omg.CORBA.BAD_PARAM {
        throw new org.omg.CORBA.NO_IMPLEMENT(); }

    // Value factory operations
    public org.omg.CORBA.portable.ValueFactory
    register_value_factory(
        String id,
        org.omg.CORBA.portable.ValueFactory factory){
        throw new org.omg.CORBA.NO_IMPLEMENT(); }
    public void unregister_value_factory(String id) {
        throw new org.omg.CORBA.NO_IMPLEMENT(); }
    public org.omg.CORBA.portable.ValueFactory
    lookup_value_factory(String id) {
        throw new org.omg.CORBA.NO_IMPLEMENT(); }
    public void set_delegate(java.lang.Object wrapper) {
        throw new org.omg.CORBA.NO_IMPLEMENT(); }
}

```

---

## ORB::create\_abstract\_interface\_tc()

Returns a pointer to a new [TypeCode](#) of kind tk\_abstract\_interface representing an IDL abstract interface.

### Parameters

id                    The repository ID that globally identifies the [TypeCode](#) object.  
name                  The simple name identifying the [TypeCode](#) object within its enclosing scope.

### See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

## ORB::create\_alias\_tc()

```
// Java
public abstract TypeCode create_alias_tc(
    String id,
    String name,
    TypeCode original_type
);
```

Returns a pointer to a new [TypeCode](#) of kind tk\_alias representing an IDL alias.

### Parameters

id                    The repository ID that globally identifies the [TypeCode](#) object.  
name                  The simple name identifying the [TypeCode](#) object within its enclosing scope.  
original\_type        A pointer to the actual [TypeCode](#) object this alias represents.

### See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

---

## ORB.create\_any()

```
// Java
public abstract Any create_any();
```

Creates a new empty Any.

## ORB::create\_array\_tc()

```
// Java
public abstract TypeCode create_array_tc(
    int length,
    TypeCode element_type
);
```

Returns a pointer to a new [TypeCode](#) of kind tk\_array representing an IDL array.

### Parameters

length            The length of the array.  
element\_type     The data type for the elements of the array.

### See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

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

```
void create_context_list(ContextList\_out list);
```

Creates an empty [ContextList](#) object for use with a DII request. You can add context strings to the list using [ContextList::add\(\)](#) and then pass the list as a parameter to [Object::\\_create\\_request\(\)](#).

### Parameters

list              A reference to the new [ContextList](#).

### See Also

[CORBA::ContextList](#)  
[CORBA::Object::\\_create\\_request\(\)](#)

---

## ORB::create\_enum\_tc()

```
// Java
public abstract TypeCode create_enum_tc(
    String id,
    String name,
    EnumMember[] members
);
```

Returns a pointer to a new [TypeCode](#) of kind tk\_enum representing an IDL enumeration.

### Parameters

id	The repository ID that globally identifies the <a href="#">TypeCode</a> object.
name	The simple name identifying the <a href="#">TypeCode</a> object within its enclosing scope.
members	The sequence of enumeration members.

### See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

## ORB::create\_environment()

```
// Java
public abstract Environment create_environment();
```

Gets a newly created [Environment](#) object.

### See Also

[CORBA::Environment](#)

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

```
// Java
public abstract ExceptionList create_exception_list();
```

Creates an empty [ExceptionList](#) object for use with a DII request. You can add user-defined exceptions to the list using [ExceptionList::add\(\)](#) and then pass the list as a parameter to [Object::\\_create\\_request\(\)](#).

### See Also

[CORBA::ExceptionList](#)  
[CORBA::Object::\\_create\\_request\(\)](#)

---

## ORB::create\_exception\_tc()

```
// Java
public abstract TypeCode create_exception_tc(
    String id,
    String name,
    StructMember[] members
);
```

Returns a pointer to a new [TypeCode](#) of kind tk\_except representing an IDL exception.

### Parameters

id	The repository ID that globally identifies the <a href="#">TypeCode</a> object.
name	The simple name identifying the <a href="#">TypeCode</a> object within its enclosing scope.
members	The sequence of members.

### See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

## ORB::create\_fixed\_tc()

Returns a pointer to a new [TypeCode](#) of kind tk\_fixed representing an IDL fixed point type.

### Parameters

digits	The number of digits for the fixed point type.
scale	The scale of the fixed point type.

### See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

## ORB::create\_interface\_tc()

```
// Java
public abstract TypeCode create_interface_tc(
    String id, String name
);
```



---

Returns a pointer to a new [TypeCode](#) representing an IDL interface.

### Parameters

`id`                   The repository ID that globally identifies the [TypeCode](#) object.  
`name`                   The simple name identifying the [TypeCode](#) object within its enclosing scope.

### See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

## ORB::create\_list()

```
// Java
public abstract NVList create_list(int count);
```

Allocates space for an empty [NVList](#) of the size specified by `count` to contain [NamedValue](#) objects. A list of [NamedValue](#) object can be used to describe arguments to a request when using the Dynamic Invocation Interface. You can add [NamedValue](#) items to list using the [NVList::add\\_item\(\)](#) routine.

### Parameters

`count`                   Number of elements anticipated for the new [NVList](#). This is a hint to help with storage allocation.

### See Also

[CORBA::NVList](#)  
[CORBA::NamedValue](#)  
[CORBA::ORB::create\\_operation\\_list\(\)](#)  
[CORBA::Request\(\)](#)

## ORB::create\_named\_value()

```
// Java
public abstract NamedValue create_named_value(
    String    name,
    Any value,
    int flags
);
```

Creates [NamedValue](#) objects you can use as return value parameters in the [Object::\\_create\\_request\(\)](#) method.

---

## Parameters

value            A pointer to the [NamedValue](#) object created. You must release the reference when it is no longer needed, or assign it to a [NamedValue\\_var](#) variable for automatic management.

## See Also

[CORBA::NVList](#)  
[CORBA::NamedValue](#)  
[CORBA::Any](#)  
[CORBA::ORB::create\\_list\(\)](#)

## ORB::create\_native\_tc()

Returns a pointer to a new [TypeCode](#) of kind `tk_native` representing an IDL native type.

## Parameters

id                The repository ID that globally identifies the [TypeCode](#) object.  
name             The simple name identifying the [TypeCode](#) object within its enclosing scope.

## See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

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

```
// Java
public abstract NVList create_operation_list(
    OperationDef operation
);
```

Creates an [NVList](#) initialized with the argument descriptions for the operation specified in `operation`.

## Parameters

operation        A pointer to the interface repository object describing the operation.

---

Each element in the list is of type [NamedValue](#) whose `value` member (of type [CORBA::Any](#)) has a valid type that denotes the type of the argument. The value of the argument is not filled in.

Use of this method requires that the relevant IDL file be compiled with the `-R` option.

**See Also**

[CORBA::NVList](#)  
[CORBA::NamedValue](#)  
[CORBA::Any](#)  
[CORBA::ORB::create\\_list\(\)](#)

### **ORB::create\_output\_stream()**

```
// Java
public abstract
    org.omg.CORBA.portable.OutputStream create_output_stream();
```

Creates a new `org.omg.CORBA.portable.OutputStream` into which IDL method parameters can be marshalled during method invocation.

### **ORB::create\_policy()**

Returns a reference to a newly created [Policy](#) object.

**Parameters**

<code>type</code>	The <a href="#">PolicyType</a> of the <a href="#">Policy</a> object to be created.
<code>value</code>	The value for the initial state of the <a href="#">Policy</a> object created.

**See Also**

[CORBA::Policy](#)  
[CORBA::PolicyType](#)  
[CORBA::PolicyErrorCode](#)

### **ORB::create\_recursive\_tc()**

Returns a pointer to a recursive [TypeCode](#), which serves as a place holder for a concrete [TypeCode](#) during the process of creating type codes that contain recursion. After the recursive [TypeCode](#) has been properly embedded in the

---

enclosing [TypeCode](#), which corresponds to the specified repository id, it will act as a normal [TypeCode](#).

### Parameters

id                    The repository ID of the enclosing type for which the recursive [TypeCode](#) is serving as a place holder.

Invoking operations on the recursive [TypeCode](#) before it has been embedded in the enclosing [TypeCode](#) will result in undefined behavior.

### Examples

The following IDL type declarations contains [TypeCode](#) recursion:

```
// IDL
struct foo {
    long value;
    sequence<foo> chain;
};

valuetype V {
    public V member;
};
```

### See Also

[CORBA::TypeCode](#)

## ORB::create\_sequence\_tc()

```
// Java
public abstract TypeCode create_sequence_tc(
    int bound,
    TypeCode element_type
);
```

Returns a pointer to a new [TypeCode](#) of kind tk\_sequence representing an IDL sequence.

### Parameters

bound                The upper bound of the sequence.

element\_type        The data type for the elements of the sequence.

### See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

---

## ORB::create\_string\_tc()

```
// Java
public abstract TypeCode create_string_tc(int bound);
```

Returns a pointer to a new [TypeCode](#) of kind `tk_string` representing an IDL string.

### Parameters

`bound`            The upper bound of the string.

### See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

## ORB::create\_struct\_tc()

```
// Java
public abstract TypeCode create_struct_tc(
    String id,
    String name,
    StructMember[] members
);
```

Returns a pointer to a new [TypeCode](#) of kind `tk_struct` representing an IDL structure.

### Parameters

`id`                The repository ID that globally identifies the [TypeCode](#) object.  
`name`             The simple name identifying the [TypeCode](#) object within its enclosing scope.  
`members`         The sequence of structure members.

### See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

## ORB::create\_union\_tc()

```
// Java
public abstract TypeCode create_union_tc(
    String id,
```

---

```
String name,  
TypeCode discriminator_type,  
UnionMember[] members  
);
```

Returns a pointer to a [TypeCode](#) of kind tk\_union representing an IDL union.

### Parameters

id	The repository ID that globally identifies the <a href="#">TypeCode</a> object.
name	The simple name identifying the <a href="#">TypeCode</a> object within its enclosing scope.
discriminator_type	The union discriminator type.
members	The sequence of union members.

### See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

## ORB::create\_value\_box\_tc()

Returns a pointer to a new [TypeCode](#) of kind tk\_value\_box representing an IDL boxed value.

### Parameters

id	The repository ID that globally identifies the <a href="#">TypeCode</a> object.
name	The simple name identifying the <a href="#">TypeCode</a> object within its enclosing scope.
original_type	A pointer to the original <a href="#">TypeCode</a> object this boxed value represents.

### See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

## ORB::create\_value\_tc()

Returns a pointer to a [TypeCode](#) of kind tk\_value representing an IDL value type.

---

## Parameters

<code>id</code>	The repository ID that globally identifies the <a href="#">TypeCode</a> object.
<code>name</code>	The simple name identifying the <a href="#">TypeCode</a> object within its enclosing scope.
<code>type_modifier</code>	A value type modifier.
<code>concrete_base</code>	A <a href="#">TypeCode</a> for the immediate concrete value type base of the value type for which the <a href="#">TypeCode</a> is being created. If the value type does not have a concrete base, use a nil <a href="#">TypeCode</a> reference.
<code>members</code>	The sequence of value type members.

## See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

## ORB::create\_wstring\_tc()

```
// Java  
public abstract TypeCode create_wstring_tc(int bound);
```

Returns a pointer to a new [TypeCode](#) of kind `tk_wstring` representing an IDL wide string.

## Parameters

`bound`            The upper bound of the string.

## See Also

[CORBA::TypeCode](#)  
[CORBA::TCKind](#)

## ORB::destroy()

```
void destroy();
```

This thread operation destroys the ORB so that its resources can be reclaimed by the application.

---

If `destroy()` is called on an ORB that has not been shut down (see [shutdown\(\)](#)) it will start the shut down process and block until the ORB has shut down before it destroys the ORB. For maximum portability and to avoid resource leaks, applications should always call [shutdown\(\)](#) and `destroy()` on all ORB instances before exiting.

After an ORB is destroyed, another call to [ORB\\_init\(\)](#) with the same ORB ID will return a reference to a newly constructed ORB.

## Exceptions

`BAD_INV_ORDER`, An application calls `destroy()` in a thread that is currently servicing an invocation because blocking would result in a deadlock.

`OBJECT_NOT_EXI` An operation is invoked on a destroyed ORB reference.  
ST

The exception is raised if

## See Also

[CORBA::ORB::run\(\)](#)  
[CORBA::ORB::shutdown\(\)](#)

## ORB::\_duplicate()

Returns a new reference to `obj` and increments the reference count of the object. Because object references are opaque and ORB-dependent, it is not possible for your application to allocate storage for them. Therefore, if more than one copy of an object reference is needed, use this method to create a duplicate.

## ORB::get\_default\_context()

```
// Java
public abstract Context get_default_context();
```

Obtains a `CORBA::Context` object representing the default context of the process.

## Parameters

`context`            The default context of the process.



---

**See Also**

[CORBA::Context](#)  
[CORBA::NVList](#)

## ORB::get\_next\_response()

```
// Java  
public abstract Request get_next_response();
```

Gets the next response for a request that has been sent.

You can call `get_next_response()` successively to determine the outcomes of the individual requests from [send\\_multiple\\_requests\\_deferred\(\)](#) calls. The order in which responses are returned is not necessarily related to the order in which the requests are completed.

**Exceptions**

`WrongTransaction` The thread invoking this method has a non-null transaction context that differs from that of the request *and* the request has an associated transaction context.

**See Also**

[CORBA::ORB::send\\_multiple\\_requests\\_deferred\(\)](#)  
[CORBA::Request::get\\_response\(\)](#)  
[CORBA::Request::send\\_deferred\(\)](#)  
[CORBA::ORB::poll\\_next\\_response\(\)](#)

## ORB::get\_primitive\_tc()

```
// Java  
public abstract TypeCode get_primitive_tc(TCKind tcKind);
```

Retrieves the `TypeCode` object that represents the given primitive IDL type.

## ORB.init()

```
public static ORB init(Strings[] args, Properties props);
```

Creates a new ORB instance for a standalone application.

```
public static ORB init(Applet app, Properties props);
```

Creates a new ORB instance for an applet.

---

```
public static ORB init();
```

Returns the ORB singleton object.

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

```
public abstract String[] list_initial_services();
```

Returns a sequence of [ObjectID](#) strings, each of which names a service provided by Orbix. This method allows your application to determine which objects have references available. Before you can use some services such as the naming service in your application you have to first obtain an object reference to the service.

The [ObjectIDList](#) may include the following names:

```
DynAnyFactory  
IT_Configuration  
InterfaceRepository  
NameService  
ORBPolicyManager  
POACurrent  
PSS  
RootPOA  
SecurityCurrent  
TradingService  
TransactionCurrent
```

**See Also** [CORBA::ORB::resolve\\_initial\\_references\(\)](#)

### **ORB::lookup\_value\_factory()**

Returns a pointer to the factory method.

#### **Parameters**

`id`                    A repository ID that identifies a value type factory method.

Your application assumes ownership of the returned reference to the factory. When you are done with the factory, invoke [ValueFactoryBase::\\_remove\\_ref\(\)](#) once on that factory.

**See Also** [CORBA::ValueFactory](#)  
[CORBA::ORB::register\\_value\\_factory\(\)](#)

---

[CORBA::ORB::unregister\\_value\\_factory\(\)](#)

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

```
public abstract String object_to_string(  
    org.omg.CORBA.Object obj  
);
```

Returns a string representation of an object reference. An object reference can be translated into a string by this method and the resulting value stored or communicated in whatever ways strings are manipulated.

### Parameters

`obj`                      Object reference to be translated to a string.

Use [string\\_to\\_object\(\)](#) to translate the string back to the corresponding object reference.

A string representation of an object reference has the prefix `IOR:` followed by a series of hexadecimal octets. The hexadecimal strings are generated by first turning an object reference into an *interoperable object reference* (IOR), and then encapsulating the IOR using the encoding rules of *common data representation* (CDR). The content of the encapsulated IOR is then turned into hexadecimal digit pairs, starting with the first octet in the encapsulation and going until the end. The high four bits of each octet are encoded as a hexadecimal digit, then the low four bits are encoded.

---

**Note:** Because an object reference is opaque and may differ from ORB to ORB, the object reference itself is not a convenient value for storing references to objects in persistent storage or communicating references by means other than invocation.

---

### See Also

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

---

## ORB::perform\_work()

```
void perform_work();
```

A thread function that provides execution resources to your application if called by the main thread. This function does nothing if called by any other thread.

### Exceptions

You can use `perform_work()` and `work_pending()` for a simple polling loop that multiplexes the main thread among the ORB and other activities. Such a loop would most likely be used in a single-threaded server. A multi-threaded server would need a polling loop only if there were both ORB and other code that required use of the main thread.

### See Also

[CORBA::ORB::run\(\)](#)  
[CORBA::ORB::work\\_pending\(\)](#)

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

```
public abstract boolean poll_next_response();
```

Returns 1 (true) if any request has completed or returns 0 (false) if none have completed. This method returns immediately, whether any request has completed or not.

You can call this method successively to determine whether the individual requests specified in a [send\\_multiple\\_requests\\_oneway\(\)](#) or [send\\_multiple\\_requests\\_deferred\(\)](#) call have completed successfully.

Alternatively you can call [Request::poll\\_response\(\)](#) on the individual [Request](#) objects in the sequence of requests passed to [send\\_multiple\\_requests\\_oneway\(\)](#) or [send\\_multiple\\_requests\\_deferred\(\)](#).

### See Also

[CORBA::ORB::get\\_next\\_response\(\)](#)  
[CORBA::ORB::send\\_multiple\\_requests\\_oneway\(\)](#)  
[CORBA::ORB::send\\_multiple\\_requests\\_deferred\(\)](#)  
[CORBA::Request::poll\\_response\(\)](#)

## ORB::register\_value\_factory()

Registers a value type factory method with the ORB for a particular value type. The method returns a null pointer if no previous factory was registered for the type. If a factory is already registered for the value type, the method replaces

---

the factory and returns a pointer to the previous factory for which the caller assumes ownership.

### Parameters

`id` A repository ID that identifies the factory.

`factory` The application-specific factory method that the ORB calls whenever it needs to create the value type during the unmarshaling of value instances.

When a value type factory is registered with the ORB, the ORB invokes [ValueFactoryBase::\\_add\\_ref\(\)](#) once on the factory before returning from [register\\_value\\_factory\(\)](#). When the ORB is done using that factory, the reference count is decremented once with [ValueFactoryBase::\\_remove\\_ref\(\)](#). This can occur in any of the following circumstances:

- If the factory is explicitly unregistered via [unregister\\_value\\_factory\(\)](#), the ORB invokes [ValueFactoryBase::\\_remove\\_ref\(\)](#) once on the factory.
- If the factory is implicitly unregistered due to a call to [shutdown\(\)](#), the ORB invokes [ValueFactoryBase::\\_remove\\_ref\(\)](#) once on each registered factory.
- If you replace a factory by calling this [register\\_value\\_factory\(\)](#) again, you should invoke [ValueFactoryBase::\\_remove\\_ref\(\)](#) once on the previous factory.

### See Also

[CORBA::ValueFactory](#)  
[CORBA::ORB::lookup\\_value\\_factory\(\)](#)  
[CORBA::ORB::unregister\\_value\\_factory\(\)](#)

### ORB::resolve\_initial\_references()

```
public abstract org.omg.CORBA.Object  
    resolve_initial_references(String object_name)  
        throws org.omg.CORBA.ORBPackage.InvalidName;
```

Returns an object reference for a desired service.

---

## Parameters

`id` The name of the desired service. Use [list\\_initial\\_services\(\)](#) to obtain the list of services supported.

Applications require a portable means by which to obtain some initial object references such as the root POA, the interface repository, and various object services instances. The functionality of `resolve_initial_references()` and [list\\_initial\\_services\(\)](#) is like a simplified, local version of the naming service that has only a small set of objects in a flattened single-level name space.

The object reference returned must be narrowed to the correct object type. For example, the object reference returned from resolving the `id` name `InterfaceRepository` must be narrowed to the type `CORBA::Repository`.

## See Also

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

## ORB::run()

```
void run();
```

A thread method that enables the ORB to perform work using the main thread. If called by any thread other than the main thread, this method simply waits until the ORB has shut down.

This method provides execution resources to the ORB so that it can perform its internal functions. Single threaded ORB implementations, and some multi-threaded ORB implementations need to use the main thread. For maximum portability, your applications should call either `run()` or [perform\\_work\(\)](#) on the main thread.

`run()` returns after the ORB has completed the shutdown process, initiated when some thread calls [shutdown\(\)](#).

## See Also

[CORBA::ORB::perform\\_work\(\)](#)

[CORBA::ORB::work\\_pending\(\)](#)

[CORBA::ORB::shutdown\(\)](#)

[CORBA::ORB::destroy\(\)](#)

---

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

```
public abstract void send_multiple_requests_deferred(  
    Request[] req  
);
```

Initiates a number of requests in parallel.

### Parameters

req                    A sequence of requests.

The method does not wait for the requests to finish before returning to the caller. The caller can use [get\\_next\\_response\(\)](#) or [Request::get\\_response\(\)](#) to determine the outcome of the requests. Memory leakage will result if one of these methods is not called for a request issued with [send\\_multiple\\_requests\\_oneway\(\)](#) or [Request::send\\_deferred\(\)](#).

### See Also

[CORBA::ORB::send\\_multiple\\_requests\\_oneway\(\)](#)  
[CORBA::Request::get\\_response\(\)](#)  
[CORBA::Request::send\\_deferred\(\)](#)  
[CORBA::ORB::get\\_next\\_response\(\)](#)

## ORB::send\_multiple\_requests\_oneway()

```
public abstract void send_multiple_requests_oneway(Request[] req);
```

Initiates a number of requests in parallel. It does not wait for the requests to finish before returning to the caller.

### Parameters

req                    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.

### See Also

[CORBA::Request::send\\_oneway\(\)](#)  
[CORBA::ORB::send\\_multiple\\_requests\\_deferred\(\)](#)

---

## ORB::shutdown()

```
void shutdown(  
    boolean wait_for_completion  
);
```

This thread method instructs the ORB to shut down in preparation for ORB destruction.

### Parameters

`wait_for_completion` Designates whether or not to wait for completion before continuing.

If the value is 1 (true), this method blocks until all ORB processing has completed, including request processing and object deactivation or other methods associated with object adapters.

If the value is 0 (false), then shut down may not have completed upon return of the method.

While the ORB is in the process of shutting down, the ORB operates as normal, servicing incoming and outgoing requests until all requests have been completed. Shutting down the ORB causes all object adapters to be shut down because they cannot exist without an ORB.

An application may also invoke [ORB::destroy\(\)](#) on the ORB itself. Invoking any other method raises exception `BAD_INV_ORDER` system with the OMG minor code 4.

### Exceptions

`BAD_INV_ORDER`, An application calls this method in a thread that is currently servicing an invocation because blocking would result in a deadlock.  
minor code 3

### See Also

[CORBA::ORB::run\(\)](#)  
[CORBA::ORB::destroy\(\)](#)



---

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

```
public abstract org.omg.CORBA.Object string_to_object(String str);
```

Returns an object reference by converting a string representation of an object reference.

### Parameters

`obj_ref_string` String representation of an object reference to be converted.

To guarantee that an ORB will understand the string form of an object reference, the string must have been produced by a call to [object\\_to\\_string\(\)](#).

### See Also

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

## ORB::unregister\_value\_factory()

Unregisters a value type factory method from the ORB.

### Parameters

`id` A repository ID that identifies a value type factory method.

### See Also

[CORBA::ValueFactory](#)

[CORBA::ORB::lookup\\_value\\_factory\(\)](#)

[CORBA::ORB::register\\_value\\_factory\(\)](#)

## ORB::work\_pending()

```
boolean work_pending();
```

This thread method returns an indication of whether the ORB needs the main thread to perform some work. A return value of 1 (true) indicates that the ORB needs the main thread to perform some work and a return value of 0 (false) indicates that the ORB does not need the main thread.

### Exceptions

`BAD_INV_ORDER`, The method is called after the ORB has shutdown.  
minor code 4

### See Also

[CORBA::ORB::run\(\)](#)

[CORBA::ORB::perform\\_work\(\)](#)



# CORBA::Policy Interface

An ORB or CORBA service may choose to allow access to certain choices that affect its operation. This information is accessed in a structured manner using interfaces derived from the `Policy` interface defined in the CORBA module. A CORBA service is not required to use this method of accessing operating options, but may choose to do so.

This chapter is divided into the following sections:

- “Quality of Service Framework”
- “Policy Methods”

The following policies are available. These are classes that inherit from the `CORBA::Policy` class:

**Table 6:** *Policies*

Category	Policy
CORBA and IT_CORBA	<code>IT_CORBA::WellKnownAddressingPolicy</code>
PortableServer and IT_PortableServer	<code>PortableServer::ThreadPolicy</code> <code>PortableServer::LifespanPolicy</code> <code>PortableServer::IdUniquenessPolicy</code> <code>PortableServer::IdAssignmentPolicy</code> <code>PortableServer::ImplicitActivationPolicy</code> <code>PortableServer::ServantRetentionPolicy</code> <code>PortableServer::RequestProcessingPolicy</code> <code>IT_PortableServer::ObjectDeactivationPolicy</code> <code>IT_PortableServer::PersistenceModePolicy</code>

You create instances of a policy by calling `CORBA::ORB::create_policy()`.

---

## Quality of Service Framework

A `Policy` is the key component for a standard *Quality of Service framework* (QoS). In this framework, all qualities are defined as interfaces derived from `CORBA::Policy`. This framework is how all service-specific qualities are defined. The components of the framework include:

<a href="#">Policy</a>	This base interface from which all QoS objects derive.
<a href="#">PolicyList</a>	A sequence of <code>Policy</code> objects.
<a href="#">PolicyManager</a>	An interface with operations for querying and overriding QoS policy settings.
Policy Transport Mechanisms	Mechanisms for transporting policy values as part of interoperable object references and within requests. These include: <ul style="list-style-type: none"><li>• <code>TAG POLICIES</code> - A Profile Component containing the sequence of QoS policies exported with the object reference by an object adapter.</li><li>• <code>INVOCATION POLICIES</code> - A Service Context containing a sequence of QoS policies in effect for the invocation.</li></ul>

Most policies are appropriate only for management at either the server or client, but not both. Server-side policies are associated with a POA. Client-side policies are divided into ORB-level, thread-level, and object-level policies. At the thread and ORB levels, use the [PolicyManager](#) interface to query the current set of policies and override these settings.

### POA Policies for Servers

Server-side policy management is handled by associating QoS `Policy` objects with a POA. Since all QoS are derived from interface `Policy`, those that are applicable to server-side behavior can be passed as arguments to `POA::create_POA()`. Any such policies that affect the behavior of requests (and therefore must be accessible by the ORB at the client side) are exported within the object references that the POA creates. It is clearly noted in a POA policy definition when that policy is of interest to the client. For those policies

that can be exported within an object reference, the absence of a value for that policy type implies that the target supports any legal value of that [PolicyType](#).

## ORB-level Policies for Clients

You obtained the ORB's locality-constrained [PolicyManager](#) through an invocation of `CORBA::ORB::resolve_initial_references()`, specifying an identifier of `ORBPolicyManager`. This [PolicyManager](#) has operations through which a set of policies can be applied and the current overriding policy settings can be obtained. Policies applied at the ORB level override any system defaults.

## Thread-level Policies for Clients

You obtained a thread's locality-constrained [PolicyCurrent](#) through an invocation of `CORBA::ORB::resolve_initial_references()`, specifying an identifier of [PolicyCurrent](#). Policies applied at the thread-level override any system defaults or values set at the ORB level. When accessed from a newly spawned thread, the [PolicyCurrent](#) initially has no overridden policies. The [PolicyCurrent](#) also has no overridden values when a POA with `ThreadPolicy` of `ORB_CONTROL_MODEL` dispatches an invocation to a servant. Each time an invocation is dispatched through a `SINGLE_THREAD_MODEL` POA, the thread-level overrides are reset to have no overridden values.

## Object-level Policies for Clients

Operations are defined on the base [Object](#) interface through which a set of policies can be applied. Policies applied at the object level override any system defaults or values set at the ORB or thread levels. In addition, accessors are defined for querying the current overriding policies set at the object level, and for obtaining the current effective client-side policy of a given [PolicyType](#). The effective client-side policy is the value of a [PolicyType](#) that would be in effect if a request were made. This is determined by checking for overrides at the object level, then at the thread level, and finally at the ORB level. If no overriding policies are set at any

---

level, the system-dependent default value is returned. Portable applications are expected to override the ORB-level policies since default values are not specified in most cases.

## Policy Methods

The `Policy` interface is as follows:

```
// IDL in module CORBA
interface Policy {
    readonly attribute PolicyType policy\_type;
    Policy copy\(\);
    void destroy\(\);
};
```

### Policy::policy\_type Attribute

```
// IDL
readonly attribute PolicyType policy_type;

// Java
public int policy_type();
```

This read-only attribute returns the constant value of type [PolicyType](#) that corresponds to the type of the `Policy` object.

### Policy::copy()

```
// IDL
Policy copy();

// Java
org.omg.CORBA.Policy copy();
```

This operation copies the `Policy` object. The copy does not retain any relationships that the original policy had with any domain, or object.

### **Policy::destroy()**

```
// IDL
void destroy();

// Java
public void destroy();
```

This operation destroys the `Policy` object. It is the responsibility of the `Policy` object to determine whether it can be destroyed.

**Enhancement** Orbix guarantees to always destroy all local objects it creates when the last reference to them is released so you do not have to call `destroy()`. However, code that relies on this feature is not strictly CORBA compliant and may leak resources with other ORBs.

### **Exceptions**

`NO_PERMISSION` The policy object determines that it cannot be destroyed.





# CORBA::PolicyCurrent Class

The `PolicyCurrent` interface allows access to policy settings at the current programming context level. Within a client, you obtain a `PolicyCurrent` object reference to set the quality of service for all invocations in the current thread. You obtain a reference to this interface by invoking [ORB::resolve\\_initial\\_references\(\)](#).

The `PolicyCurrent` interface is derived from the [PolicyManager](#) and the [Current](#) interfaces. The [PolicyManager](#) interface allows you to change the policies for each invocation and the [Current](#) interface allows control from the current thread.

Policies applied at the thread level override any system defaults or values set at the ORB level. When accessed from a newly spawned thread, the `PolicyCurrent` initially has no overridden policies. The `PolicyCurrent` also has no overridden values when a POA with `ThreadPolicy` of `ORB_CONTROL_MODEL` dispatches an invocation to a servant. Each time an invocation is dispatched through a POA of the `SINGLE_THREAD_MODEL`, the thread-level overrides are reset to have no overridden values.

```
// Java
package org.omg.CORBA;
public interface PolicyCurrent extends
    org.omg.CORBA.PolicyManager,
    org.omg.CORBA.Current {}
```



# CORBA::PolicyManager Class

`PolicyManager` is an interface with operations for querying and overriding QoS policy settings. It includes mechanisms for obtaining policy override management operations at each relevant application scope. You obtain the ORB's `PolicyManager` by invoking `ORB::resolve_initial_references()` with the `ObjectId` `ORBPolicyManager`.

You use a `CORBA::PolicyCurrent` object, derived from `CORBA::Current`, for managing the thread's QoS policies. You obtain a reference to this interface by invoking `ORB::resolve_initial_references()` with the `ObjectId` `PolicyCurrent`.

- Accessor operations on `CORBA::Object` allow querying and overriding of QoS at the object reference scope.
- The application of QoS on a POA is done through the currently existing mechanism of passing a `PolicyList` to `POA::create_POA()`.

## `PolicyManager::get_policy_overrides()`

```
// Java
org.omg.CORBA.Policy[] get_policy_overrides( int[] ts );
```

### Parameters

Returns a list containing the overridden policies for the requested policy types. This returns only those policy overrides that have been set at the specific scope corresponding to the target `PolicyManager` (no evaluation is done with respect to overrides at other scopes). If none of the requested policy types are overridden at the target `PolicyManager`, an empty sequence is returned.

### Parameters

`ts`                    A sequence of policy types to get. If the specified sequence is empty, the method returns all policy overrides at this scope.

### See Also

[CORBA::PolicyManager::set\\_policy\\_overrides\(\)](#)

---

## PolicyManager::set\_policy\_overrides()

```
// Java
void set_policy_overrides(
    org.omg.CORBA.Policy[] policies,
    org.omg.CORBA.SetOverrideType set_add
) throws org.omg.CORBA.InvalidPolicies;
```

Modifies the current set of overrides with the requested list of policy overrides.

### Parameters

- |                       |   |
|-----------------------|---|
| <code>policies</code> | A sequence of references to policy objects.   |
| <code>set_add</code>  | Indicates whether the policies in the <code>policies</code> parameter should be added to existing overrides in the <code>PolicyManager</code> or used to replace existing overrides: <ul style="list-style-type: none"><li>• Use <a href="#">ADD_OVERRIDE</a> to add policies onto any other overrides that already exist in the <code>PolicyManager</code>.</li><li>• Use <a href="#">SET_OVERRIDE</a> to create a clean <code>PolicyManager</code> free of any other overrides.</li></ul> |

Invoking the method with an empty sequence of policies and a mode of [SET\\_OVERRIDE](#) removes all overrides from a `PolicyManager`.

There is no evaluation of compatibility with policies set within other policy managers.

### Exceptions

- |  |  |
|--|--|
| <code>NO_PERMISSION</code>             | Only certain policies that pertain to the invocation of an operation at the client end can be overridden using this operation. This exception is raised if you attempt to override any other policy. |
| <a href="#">InvalidPolicyException</a> | The request would put the set of overriding policies for the target <code>PolicyManager</code> in an inconsistent state. No policies are changed or added.   |

# CORBA::PrimitiveDef Interface

Interface `PrimitiveDef` represents an IDL primitive type such as `short`, `long`, and others. `PrimitiveDef` objects are anonymous (unnamed) and owned by the interface repository.

Objects of type `PrimitiveDef` cannot be created directly. You can obtain a reference to a `PrimitiveDef` by calling `Repository::get_primitive()`.

```
// IDL in module CORBA.
interface PrimitiveDef: IDLType {
    readonly attribute PrimitiveKind kind;
};
```

## See Also

[CORBA::PrimitiveKind](#)  
[CORBA::IDLType](#)  
[CORBA::Repository::get\\_primitive\(\)](#)

## PrimitiveDef::kind Attribute

```
// IDL
readonly attribute PrimitiveKind kind;

// Java
org.omg.CORBA.PrimitiveKind kind();
```

Identifies which of the IDL 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](#).

## See Also

[CORBA::IDLType](#)  
[CORBA::Object](#)  
[CORBA::StringDef](#)



# CORBA::Repository Interface

The interface repository itself is a container for IDL type definitions. Each interface repository is represented by a global root `Repository` object.

The `Repository` interface describes the top-level object for a repository name space. It contains definitions of constants, typedefs, exceptions, interfaces, value types, value boxes, native types, and modules.

You can use the `Repository` operations to look up any IDL definition, by either name or identity, that is defined in the global name space, in a module, or in an interface. You can also use other `Repository` operations to create information for the interface repository. See [Table 7](#):

**Table 7:** *Operations of the Repository Interface*

Read Operations	Write Operations
<a href="#">get_canonical_typecode()</a>	<a href="#">create_array()</a>
<a href="#">get_primitive()</a>	<a href="#">create_fixed()</a>
<a href="#">lookup_id()</a>	<a href="#">create_sequence()</a>
	<a href="#">create_string()</a>
	<a href="#">create_wstring()</a>

The five `create_type` operations create new interface repository objects defining anonymous types. Each anonymous type definition must be used in defining exactly one other object. Because the interfaces for these anonymous types are not derived from [Contained](#), it is your responsibility to invoke in your application `destroy()` on the returned object if it is not successfully used in creating a definition that is derived from [Contained](#).

The `Repository` interface is as follows:

```
// IDL in module CORBA.
interface Repository : Container {
    Contained lookup\_id(
        in RepositoryId search_id
    );
    TypeCode get\_canonical\_typecode(
        in TypeCode tc
```

---

```

    );
    PrimitiveDef get\_primitive(
        in PrimitiveKind kind
    );
    StringDef create\_string(
        in unsigned long bound
    );
    WstringDef create\_wstring(
        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
    );
    FixedDef create\_fixed(
        in unsigned short digits,
        in short scale
    );
};

```

Note that although a `Repository` does not have a `RepositoryId` associated with it (because it derives only from [Container](#) and not from [Contained](#)) you can assume that its default `RepositoryId` is an empty string. This allows a value to be assigned to the `defined_in` field of each description structure for [ModuleDef](#), [InterfaceDef](#), [ValueDef](#), [ValueBoxDef](#), [TypedefDef](#), [ExceptionDef](#) and [ConstantDef](#) that may be contained immediately within a `Repository` object.

#### See Also

[CORBA::Container](#)

### Repository::create\_array()

```

// IDL
ArrayDef create_array(
    in unsigned long length,
    in IDLType element_type
);

```



---

```
// Java
org.omg.CORBA.ArrayDef create_array(
    int length,
    org.omg.CORBA.IDLType element_type
);
```

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. If the created object is not successfully used in the definition of a [Contained](#) object, it is your application's responsibility to delete it.

### Parameters

length            The number of elements in the array.  
element\_type    The type of element that the array will contain.

### See Also

[CORBA::ArrayDef](#)  
[CORBA::IObject](#)

## Repository::create\_fixed()

```
// IDL
FixedDef create_fixed (
    in unsigned short digits,
    in short scale
);

// Java
org.omg.CORBA.FixedDef create_fixed(
    short digits,
    short scale
);
```

Returns a new fixed-point object defining an anonymous (unnamed) type. The new object must be used in the definition of exactly one other object. It is deleted when the object it is contained in is deleted. If the created object is not successfully used in the definition of a [Contained](#) object, it is your application's responsibility to delete it.

---

## Parameters

digits	The number of digits in the fixed-point number. Valid values must be between 1 and 31, inclusive.
scale	The scale.

## Repository::create\_sequence()

```
// IDL
SequenceDef create_sequence (
    in unsigned long bound,
    in IDLType element_type
);

// Java
org.omg.CORBA.SequenceDef create_sequence(
    int bound,
    org.omg.CORBA.IDLType element_type
);
```

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. If the created object is not successfully used in the definition of a [Contained](#) object, it is your application's responsibility to delete it.

## 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 will contain.

## See Also

[CORBA::SequenceDef](#)

## Repository::create\_string()

```
// IDL
StringDef create_string(
    in unsigned long bound
);
```

---

```
// Java
org.omg.CORBA.StringDef create_string( int bound );
```

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. If the created object is not successfully used in the definition of a [Contained](#) object, it is your application's responsibility to delete it.

### Parameters

bound                    The maximum number of characters in the string. (This cannot be 0.)

Use [get\\_primitive\(\)](#) to create unbounded strings.

### See Also

[CORBA::StringDef](#)  
[CORBA::Repository::get\\_primitive\(\)](#)

## Repository::create\_wstring()

```
// IDL
StringDef create_wstring (
    in unsigned long bound
);

// Java
org.omg.CORBA.WstringDef create_wstring( int bound );
```

Returns a new wide string object defining an anonymous (unnamed) type. The new wide 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. If the created object is not successfully used in the definition of a [Contained](#) object, it is your application's responsibility to delete it.

### Parameters

bound                    The maximum number of characters in the string. (This cannot be 0.)

Use [get\\_primitive\(\)](#) to create unbounded strings.

### See Also

[CORBA::WstringDef](#)  
[CORBA::Repository::get\\_primitive\(\)](#)

---

## Repository::get\_canonical\_typecode()

```
// IDL
TypeCode get_canonical_typecode(
    in TypeCode tc
);

// Java
org.omg.CORBA.TypeCode get_canonical_typecode(
    org.omg.CORBA.TypeCode tc
);
```

Returns a [TypeCode](#) that is equivalent to `tc` that also includes all repository ids, names, and member names.

### Parameters

`tc`                    The [TypeCode](#) to lookup.

If the top level [TypeCode](#) does not contain a `RepositoryId` (such as array and sequence type codes or type codes from older ORBs) or if it contains a `RepositoryId` that is not found in the target `Repository`, then a new [TypeCode](#) is constructed by recursively calling [get\\_canonical\\_typecode\(\)](#) on each member [TypeCode](#) of the original [TypeCode](#).

## Repository::get\_primitive()

```
// IDL
PrimitiveDef get_primitive(
    in PrimitiveKind kind
);

// Java
org.omg.CORBA.PrimitiveDef get_primitive(
    org.omg.CORBA.PrimitiveKind kind
);
```

Returns a reference to a [PrimitiveDef](#) of the specified [PrimitiveKind](#). All [PrimitiveDef](#) objects are owned by the `Repository`, one primitive object per primitive type (for example, short, long, unsigned short, unsigned long and so on).

### Parameters

`kind`                    The kind of primitive to get.

---

**See Also**      [CORBA::PrimitiveDef](#)

## Repository::lookup\_id()

```
// IDL
Contained lookup_id(
    in RepositoryId search_id
);
// Java
org.omg.CORBA.Contained lookup_id( java.lang.String search_id );
```

Returns an object reference to a [Contained](#) object within the repository given its `RepositoryId`. If the repository does not contain a definition for the given ID, a nil object reference is returned.

### Parameters

`search_id`      The `RepositoryId` of the IDL definition to lookup.

**See Also**      [CORBA::Contained](#)



# CORBA::Request Class

This class is the key support class for 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.

Orbis allows invocations, that are instances of class `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.

You create a request using methods [Object::\\_create\\_request\(\)](#) or [Object::\\_request\(\)](#).

## See Also

[CORBA::Object::\\_request\(\)](#)  
[CORBA::Object::\\_create\\_request\(\)](#)

## Request::add\_in\_arg()

```
// Java
public abstract Any add\_in\_arg\(\);
```

Returns an `any` value for the input argument that is added.

## See Also

[CORBA::Request::arguments\(\)](#)  
[CORBA::Request::add\\_inout\\_arg\(\)](#)  
[CORBA::Request::add\\_out\\_arg\(\)](#)

## Request::add\_inout\_arg()

```
// Java
public abstract Any add\_inout\_arg\(\);
```

Returns an `any` value for the in/out argument that is added.

## See Also

[CORBA::Request::arguments\(\)](#)  
[CORBA::Request::add\\_in\\_arg\(\)](#)  
[CORBA::Request::add\\_out\\_arg\(\)](#)

---

## Request::add\_named\_in\_arg()

```
// Java
public abstract Any add\_named\_in\_arg(String name);
```

## Request::add\_named\_inout\_arg()

```
// Java
public abstract Any add\_named\_inout\_arg(String name);
```

## Request::add\_named\_out\_arg()

```
// Java
public abstract Any add\_named\_out\_arg(String name);
```

## Request::add\_out\_arg()

```
// Java
public abstract Any add\_out\_arg();
```

Returns an any value for the output argument that is added.

### See Also

[CORBA::Request::arguments\(\)](#)  
[CORBA::Request::add\\_in\\_arg\(\)](#)  
[CORBA::Request::add\\_inout\\_arg\(\)](#)

## Request::arguments()

```
// Java
public abstract NVList arguments();
```

Returns the arguments to the requested operation in an [NVList](#). Ownership of the return value is maintained by the `Request` and must not be freed by the caller. You can add additional arguments to the request using the `add*_arg()` helper methods.

### See Also

[CORBA::NVList](#)  
[CORBA::Request::add\\_in\\_arg\(\)](#)  
[CORBA::Request::add\\_inout\\_arg\(\)](#)  
[CORBA::Request::add\\_out\\_arg\(\)](#)



---

## Request::contexts()

```
// Java
public abstract ContextList contexts\(\);
```

Returns a pointer to a list of contexts for the request. Ownership of the return value is maintained by the `Request` and must not be freed by the caller.

## Request::ctx()

```
// Java
public abstract Context ctx\(\);
```

Returns the [Context](#) associated with a request. Ownership of the return value is maintained by the `Request` and must not be freed by the caller.

```
// Java
public abstract void ctx(Context c);
```

Inserts a [Context](#) into a request.

### Parameters

`c`                    The context to insert with the request.

## Request::env()

```
// Java
public abstract Environment env\(\);
```

Returns the `Environment` associated with the request from which exceptions raised in DII calls can be accessed. Ownership of the return value is maintained by the `Request` and must not be freed by the caller.

### See Also

[CORBA::Environment](#)

## Request::exceptions()

```
// Java
public abstract ExceptionList exceptions\(\);
```

---

Returns a pointer to list of possible application-specific exceptions for the request. Ownership of the return value is maintained by the `Request` and must not be freed by the caller.

**See Also** [CORBA::ExceptionList](#)

## **Request::get\_response()**

```
// Java
public abstract void get\_response\(\);
```

Determines whether a request has completed successfully. It returns only when the request, invoked previously using [send\\_deferred\(\)](#), has completed.

**See Also** [CORBA::Request::result\(\)](#)  
[CORBA::Request::send\\_deferred\(\)](#)

## **Request::invoke()**

```
// Java
public abstract void invoke\(\);
```

Instructs the ORB 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 [send\\_deferred\(\)](#) and [send\\_oneway\(\)](#).

**See Also** [CORBA::Request::send\\_oneway\(\)](#)  
[CORBA::Request::send\\_deferred\(\)](#)  
[CORBA::Request::result\(\)](#)

## **Request::operation()**

```
// Java
public abstract String operation\(\);
```

Returns the operation name of the request. Ownership of the return value is maintained by the `Request` and must not be freed by the caller.

---

## Request::poll\_response()

```
// Java
public abstract boolean poll\_response\(\);
```

Returns 1 (true) if the operation has completed successfully and indicates that the return value and out and inout parameters in the request are valid. Returns 0 (false) otherwise. The method returns immediately.

If your application makes an operation request using [send\\_deferred\(\)](#), it can call [poll\\_response\(\)](#) to determine whether the operation has completed. If the operation has completed, you can get the result by calling [Request::result\(\)](#).

### See Also

[CORBA::Request::send\\_deferred\(\)](#)  
[CORBA::Request::get\\_response\(\)](#)  
[CORBA::Request::result\(\)](#)

## Request::result()

```
// Java
public abstract NamedValue result\(\);
```

Returns the result of the operation request in a [NamedValue](#). Ownership of the return value is maintained by the `Request` and must not be freed by the caller.

## Request::return\_value()

```
// Java
public abstract Any return\_value\(\);
```

Returns an `any` value for the returned value of the operation.

## Request::send\_deferred()

```
// Java
public abstract void send\_deferred\(\);
```

Instructs the ORB 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.

---

To make a blocking request, use [invoke\(\)](#). You can use [poll\\_response\(\)](#) to determine whether the operation completed.

#### See Also

[CORBA::Request::send\\_oneway\(\)](#)  
[CORBA::ORB::send\\_multiple\\_requests\\_deferred\(\)](#)  
[CORBA::Request::invoke\(\)](#)  
[CORBA::Request::poll\\_response\(\)](#)  
[CORBA::Request::get\\_response\(\)](#)

### Request::send\_oneway()

```
// Java  
public abstract void send\_oneway\(\);
```

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.

You can use this method even if the operation has not been defined to be oneway in its IDL definition, however, do not expect any output or inout parameters to be updated.

To make a blocking request, use [invoke\(\)](#).

#### See Also

[CORBA::Request::send\\_deferred\(\)](#)  
[CORBA::ORB::send\\_multiple\\_requests\\_oneway\(\)](#)  
[CORBA::Request::invoke\(\)](#)  
[CORBA::Request::poll\\_response\(\)](#)  
[CORBA::Request::get\\_response\(\)](#)

### Request::set\_return\_type()

```
// Java  
public abstract void set\_return\_type(TypeCode tc);
```

Sets the [TypeCode](#) associated with a `Request` object. When using the DII with the Internet Inter-ORB Protocol (IIOP), you must set the return type of a request before invoking the request.

#### Parameters

`tc`                    The [TypeCode](#) for the return type of the operation associated with the `Request` object.

---

## Request::target()

```
// Java  
public abstract Object target\(\);
```

Gets the target object of the `Request`. Ownership of the return value is maintained by the `Request` and must not be freed by the caller.



# CORBA::SequenceDef Interface

Interface `SequenceDef` represents an IDL sequence definition in the interface repository. It inherits from the interface [IDLType](#).

```
// IDL in module CORBA.  
interface SequenceDef : IDLType {  
    attribute unsigned long bound;  
    readonly attribute TypeCode element\_type;  
    attribute IDLType element\_type\_def;  
};
```

The inherited [type](#) attribute is also described.

## See Also

[CORBA::IDLType](#)  
[CORBA::Repository::create\\_sequence\(\)](#)

## SequenceDef::bound Attribute

```
// IDL  
attribute unsigned long bound;  
  
// Java  
int bound();  
void bound( int _val );
```

The maximum number of elements in the sequence. A bound of 0 indicates an unbounded sequence.

Changing the `bound` attribute will also update the inherited `type` attribute.

## See Also

[CORBA::SequenceDef::type](#)

## SequenceDef::element\_type Attribute

```
// IDL  
readonly attribute TypeCode element_type;  
  
// Java  
org.omg.CORBA.TypeCode element_type();
```

---

The type of element contained within this sequence. The attribute `element_type_def` contains the same information.

**See Also** [CORBA::SequenceDef::element\\_type\\_def](#)

## SequenceDef::element\_type\_def Attribute

```
// IDL
attribute IDLType element_type_def;

// Java
org.omg.CORBA.IDLType element_type_def();
void element_type_def( org.omg.CORBA.IDLType _val );
```

Describes the type of element contained within this sequence. The attribute `element_type` contains the same information. Setting the `element_type_def` attribute also updates the `element_type` and [IDLType::type](#) attributes.

**See Also** [CORBA::SequenceDef::element\\_type](#)  
[CORBA::IDLType::type](#)

## SequenceDef::type Attribute

```
// IDL
readonly attribute TypeCode type;
```

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.

**See Also** [CORBA::SequenceDef::element\\_type\\_def](#)  
[CORBA::SequenceDef::bound](#)



# CORBA::ServerRequest Class

The object adapter dispatches an invocation to a DSI-based object implementation by calling `invoke()` on an object of the `DynamicImplementation` class. The parameter passed to this method is a `ServerRequest` object. This `ServerRequest` object contains the state of an incoming invocation for the DSI. This can be compared to how the `Request` class object is used in the DII approach for clients.

The following code is the complete class definition:

## ServerRequest::arguments()

```
// Java
public void arguments(org.omg.CORBA.NVList args)
```

Allows a redefinition of the following method to specify the values of incoming arguments:

```
PortableServer::DynamicImplementation::invoke()
```

### Parameters

`args`                      Obtains output and input arguments.

This method must be called *exactly* once in each execution of `invoke()`.

### See Also

[CORBA::ServerRequest::params\(\)](#)  
`PortableServer::DynamicImplementation::invoke()`

## ServerRequest::ctx()

```
// Java
public abstract Context ctx();
```

Returns the [Context](#) associated with the call.

If no `Context` was sent then this method returns `null`.

---

## ServerRequest::except()

```
public abstract void except(Any a);
```

The DIR may call `except()` at any time to return an exception to the client.

### Parameters

a                                   An `Any` containing the exception to be returned to the client.

The `Any` value passed to `except()` must contain either a system exception or one of the user exceptions specified in the `raises` expression of the invoked operation's IDL definition.

### See Also

"System Exceptions"  
`CORBA.Any`  
`CORBA.SystemException`

## ServerRequest::operation()

```
// Java  
public String operation()
```

### Parameters

Returns the name of the operation being invoked.

This method must be called at least once in each execution of the dynamic implementation routine, that is, in each redefinition of the method:

```
PortableServer::DynamicImplementation::invoke()
```

### See Also

[CORBA::ServerRequest::op\\_name\(\)](#)  
`PortableServer::DynamicImplementation::invoke()`

## ServerRequest::op\_name()

```
public abstract String op_name();
```

Returns the name of the operation being invoked.

---

## ServerRequest::params()

```
public abstract void params(NVList params);
```

This method marshals the parameters from the incoming `ServerRequest` into the supplied `params NVList`.

### Parameters

`params` An `NVList` describing the parameter types for the operation in the order in which they appear in the IDL specification (left to right).

It is up to the programmer to ensure that the `TypeCode` and flags (`ARG_IN`, `ARG_OUT` or `ARG_INOUT`) of each of the parameters are correct.

The Dynamic Implementation Routine (DIR) must call `params` with `params` containing `TypeCodes` and `Flags` describing the parameter types expected for the method.

After invoking `params()` the programmer uses the unmarshaled “in” and “inout” values as parameters to the method invocation.

When the invocation completes the programmer must insert the values for any out and inout parameters into the `params NVList` before returning.

If the operation has a return value you must also call “`result()`” .

For example:

```
// import org.omg.CORBA.*;
//
// simulate the set operation on the grid interface with
// the DSI
public void invoke(ServerRequest _req) {
    String _opName = _req.op_name() ;
    Any _ret = ORB.init().create_any();
    NVList _nvl = null;
    long [][]ma_a = // create new array;

    if(_opName.equals("set"))
    {
        _nvl = ORB.init().create_list(3);

        // create a new any
```

---

```

Any row = ORB.init().create_any();

// insert the TypeCode (tk_short) into the new Any
row.type(ORB.init().get_primitive_tc(TCKind.tk_short)) ;

// insert this Any into the NVList and set the Flag to in
_nvl.add_value(null, row, ARG_IN.value);

// create new Any,set TypeCode to short, insert into NVList
// with flag set to in
Any col = ORB.init().create_any();
col.type(ORB.init().get_primitive_tc(TCKind.tk_short));
_nvl.add_value(null, col, ARG_IN.value);

// create new Any,set TypeCode to long, insert into NVList
// with flag set to in
Any data = ORB.init().create_any();
data.type(ORB.init().get_primitive_tc(TCKind.tk_long));
_nvl.add_value(null, data, ARG_IN.value);

// get params() method to marshal data into _nvl
_req.params(_nvl);

// get the value of row,col from Any row,col
// and set this element in the array to the value
m_a[row.extract_short()][col.extract_short()] =
    data.extract_long() ;

return;
}
}

```

### See Also

CORBA.NVList class

### ServerRequest.result()

```
public abstract void result(Any a);
```

Use the `result()` method to specify the return value for the call.

---

## Parameters

`res` An `Any` containing the return value and type for the operation.

If the operation has a `void` result type, `result()` should be set to an `Any` whose type is `_tc_void`.

## See Also

`CORBA.Any` Class

## ServerRequest::set\_exception()

```
// Java
public void set_exception(Any any)
```

Allows (a redefinition of) `PortableServer::DynamicImplementation::invoke()` to return an exception to the caller.

## Parameters

`value` A pointer to an `Any`, which holds the exception returned to the caller.

## See Also

[CORBA::Environment\(\)](#)  
[PortableServer::DynamicImplementation::invoke\(\)](#)

## ServerRequest::set\_result()

```
// Java
public void set_result(org.omg.CORBA.Any any)
```

Allows `PortableServer::DynamicImplementation::invoke()` to return the result of an operation request in an `Any`.

## Parameters

`value` A pointer to a `Any`, which holds the result returned to the caller.

This method 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, then [set\\_exception\(\)](#) cannot be used.

## See Also

[CORBA::ServerRequest::set\\_exception\(\)](#)



# CORBA::String\_var Class

The 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 when execution goes out of scope.

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

Converts a `String_var` object to a `char*`.

### **See Also**

[CORBA::String\\_var::operator=\(\)](#)

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

Returns the proper string for use as an input parameter.

### **See Also**

[CORBA::String\\_var::out\(\)](#)

[CORBA::String\\_var::inout\(\)](#)

[CORBA::String\\_var::\\_retn\(\)](#)

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

Returns the proper string for use as an inout parameter.

### **See Also**

[CORBA::String\\_var::in\(\)](#)

[CORBA::String\\_var::out\(\)](#)

[CORBA::String\\_var::\\_retn\(\)](#)

## **String\_var::operator=() Assignment Operators**

Assignment operators allow you to assign values to a `String_var` from a `char*` or from another `String_var` type.

### Parameters

- `p` A character string to assign to the `String_var`.  
`s` A `String_var` to assign to the `String_var`.

**See Also** [CORBA::String\\_var::char\\*\(\)](#)

### String\_var::operator[]() Subscript Operators

Return the character at the given location of the string. Subscript operators allow access to the individual characters in the string.

### Parameters

- `index` The index location in the string.

### String\_var::out()

Returns the proper string for use as an output parameter.

**See Also** [CORBA::String\\_var::in\(\)](#)  
[CORBA::String\\_var::inout\(\)](#)  
[CORBA::String\\_var::\\_retn\(\)](#)

### String\_var::String\_var() Constructors

The default constructor.

Constructors that convert from a `char*` to a `String_var`.

The copy constructor.

### Parameters

- `p` The character string to convert to a `String_var`. The `String_var` assumes ownership of the parameter.  
`s` The original `String_var` that is copied.

**See Also** [CORBA::String\\_var::~~String\\_var\(\)](#)



---

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

The destructor.

**See Also** [CORBA::String\\_var::String\\_var\(\)](#)

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

Returns the proper string for use as a method's return value.

**See Also** [CORBA::String\\_var::inout\(\)](#)  
[CORBA::String\\_var::in\(\)](#)  
[CORBA::String\\_var::out\(\)](#)



# CORBA::StringDef Interface

Interface `StringDef` represents an IDL bounded string type in the interface repository. A `StringDef` object is anonymous, which means it is unnamed.

Use `Repository::create_string()` to obtain a new `StringDef`. Use `Repository::get_primitive()` for unbounded strings.

```
// IDL in module CORBA.
interface StringDef : IDLType {
    attribute unsigned long bound;
};
```

The inherited `type` attribute is also described.

## See Also

[CORBA::IDLType](#)  
[CORBA::Repository::create\\_string\(\)](#)

## StringDef::bound Attribute

```
// IDL
attribute unsigned long bound;

// Java
int bound();
void bound( int _val );
```

Specifies the maximum number of characters in the string. This cannot be zero.

## StringDef::type Attribute

```
// IDL
readonly attribute TypeCode type;
```

The `type` attribute is inherited from interface [IDLType](#). This attribute is a `tk_string` [TypeCode](#) that describes the string.

## See Also

[CORBA::IDLType::type](#)



# CORBA::StructDef Interface

Interface StructDef describes an IDL structure in the interface repository.

```
// IDL in module CORBA.  
interface StructDef : TypedefDef, Container {  
    attribute StructMemberSeq members;  
};
```

The inherited operation [describe\(\)](#) is also described.

## See Also

[CORBA::Contained](#)  
[CORBA::Container::create\\_struct\(\)](#)

## StructDef::describe()

```
// IDL  
Description describe();
```

`describe()` returns a [Contained::Description](#) structure. `describe()` is inherited from [Contained](#) (which [TypedefDef](#) inherits).

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](#).

## See Also

[CORBA::TypedefDef::describe\(\)](#)

## StructDef::members Attribute

```
// Java  
org.omg.CORBA.StructMember[] members();  
void members( org.omg.CORBA.StructMember[] _val );
```

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 will be set automatically to the [TypeCode](#) of the `type_def` field).

**See Also**

[CORBA::TypedefDef](#)

# CORBA::TypeCode Class

The class `TypeCode` is used to describe IDL type structures at runtime. A `TypeCode` is a value that represents an IDL invocation argument type or an IDL attribute type. A `TypeCode` is typically used as follows:

- In the dynamic invocation interface (DII) to indicate the type of an actual argument.
- By the interface repository to represent the type specification that is part of an OMG IDL declaration.
- To describe the data held by an *any* type.

A `TypeCode` consists of a *kind* that classifies the `TypeCode` as to whether it is a basic type, a structure, a sequence and so on. See the data type [TCKind](#) for all possible kinds of `TypeCode` objects.

A `TypeCode` may also include a sequence of parameters. 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`.

You typically obtain a `TypeCode` from the interface repository or it may be generated by the IDL compiler. You do not normally create a `TypeCode` in your code so the class contains no constructors, only methods to decompose the components of an existing `TypeCode`. However, if your application does require that you create a `TypeCode`, see the set of `create_Type_tc()` methods in the [ORB](#) class.

The class `TypeCode` contains the following methods:

**See Also**

[CORBA::TCKind](#)

## TypeCode::BadKind Exception

```
// Java
class CORBA.TypeCodePackage.BadKind
```

The `BadKind` exception is raised if a `TypeCode` member method is invoked for a kind that is not appropriate.

---

## TypeCode::Bounds Exception

```
// Java
class CORBA.TypeCodePackage.Bounds
```

The `Bounds` exception is raised if an attempt is made to use an index for a type's member that is greater than or equal to the number of members for the type.

The type of IDL constructs that have members include enumerations, structures, unions, value types, and exceptions. Some of the `TypeCode` methods return information about specific members of these IDL constructs. The first member has index value 0, the second has index value 1, and so on up to  $n-1$  where  $n$  is the count of the total number of members.

The order in which members are presented in the interface repository is the same as the order in which they appeared in the IDL specification.

This exception is not the same as the `CORBA::Bounds` exception.

### See Also

[CORBA::TypeCode::member\\_count\(\)](#)  
[CORBA::TypeCode::member\\_label\(\)](#)  
[CORBA::TypeCode::member\\_name\(\)](#)  
[CORBA::TypeCode::member\\_type\(\)](#)  
[CORBA::TypeCode::member\\_visibility\(\)](#)

## TypeCode::concrete\_base\_type()

```
// Java
public TypeCode concrete_base_type() throws BadKind {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

Returns a `TypeCode` for the concrete base if the value type represented by this `TypeCode` has a concrete base value type. Otherwise it returns a nil `TypeCode` reference. This method is valid to use only if the kind of `TypeCode` has a [TCKind](#) value of `tk_value`.

### Exceptions

[BadKind](#)      The kind of `TypeCode` is not valid for this method.



---

## TypeCode::content\_type()

```
// Java
public abstract TypeCode content_type() throws BadKind
```

For sequences and arrays this method returns a reference to the element type. For aliases it returns a reference to the original type. For a boxed value type it returns a reference to the boxed type. This method is valid to use if the kind of `TypeCode` is one of the following [TCKind](#) values:

```
tk_alias
tk_array
tk_sequence
tk_value_box
```

### Exceptions

[BadKind](#) The kind of `TypeCode` is not valid for this method.

## TypeCode::default\_index()

```
// Java
public abstract int default_index() throws BadKind;
```

Returns the index of the default union member, or -1 if there is no default member. This method is valid to use only if the kind of `TypeCode` has a [TCKind](#) value of `tk_union`.

### Exceptions

[BadKind](#) The kind of `TypeCode` is not valid for this method.

### See Also

[CORBA::TypeCode::member\\_label\(\)](#)

## TypeCode::discriminator\_type()

```
// Java
public abstract TypeCode discriminator_type() throws BadKind;
```

Returns a `TypeCode` for the union discriminator type. This method is valid to use only if the kind of `TypeCode` has a [TCKind](#) value of `tk_union`.

---

## Exceptions

[BadKind](#) The kind of `TypeCode` is not valid for this method.

## See Also

[CORBA::TypeCode::default\\_index\(\)](#)  
[CORBA::TypeCode::member\\_label\(\)](#)

## TypeCode::equal()

```
// Java
public abstract boolean equal(TypeCode tc);
```

Returns 1 (true) if this `TypeCode` and the `tc` parameter are equal. Returns 0 (false) otherwise. Two type codes are equal if the set of legal operations is the same and invoking an operation from one set returns the same results as invoking the operation from the other set.

## Parameters

`tc` The `TypeCode` to compare.

## See Also

[CORBA::TypeCode::equivalent\(\)](#)

## TypeCode::equivalent()

```
// Java
public boolean equivalent(TypeCode tc) {
    throw new org.omg.CORBA.NO_IMPLEMENT(); }
```

Returns 1 (true) if this `TypeCode` and the `tc` parameter are equivalent. Returns 0 (false) otherwise.

## Parameters

`tc` The `TypeCode` to compare.

`equivalent()` is typically used by the ORB to determine type equivalence for values stored in an IDL `any`. You can use `equal()` to compare type codes in your application. `equivalent()` would return true if used to compare a type and an alias of that type while `equal()` would return false.

## See Also

[CORBA::TypeCode::equal\(\)](#)

---

## TypeCode::fixed\_digits()

```
// Java
public short fixed_digits() throws BadKind {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

Returns the number of digits in the fixed point type. This method is valid to use only if the kind of `TypeCode` has a [TCKind](#) value of `tk_fixed`.

### Exceptions

[BadKind](#) The kind of `TypeCode` is not valid for this method.

### See Also

[CORBA::TypeCode::fixed\\_scale\(\)](#)

## TypeCode::fixed\_scale()

```
// Java
public short fixed_scale() throws BadKind {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

Returns the scale of the fixed point type. This method is valid to use only if the kind of `TypeCode` has a [TCKind](#) value of `tk_fixed`.

### Exceptions

[BadKind](#) The kind of `TypeCode` is not valid for this method.

### See Also

[CORBA::TypeCode::fixed\\_digits\(\)](#)

## TypeCode::get\_compact\_typecode()

```
// Java
public TypeCode get_compact_typecode() {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

Removes all optional name and member name fields from the `TypeCode` and returns a reference to the compact `TypeCode`. This method leaves all alias type codes intact.

---

## TypeCode::id()

```
// Java
public abstract String id() throws BadKind;
```

Returns the [RepositoryId](#) that globally identifies the type.

Type codes that always have a [RepositoryId](#). include object references, value types, boxed value types, native, and exceptions. Other type codes that also always have a [RepositoryId](#) and are obtained from the interface repository or [ORB::create\\_operation\\_list\(\)](#) include structures, unions, enumerations, and aliases. In other cases `id()` could return an empty string.

The `TypeCode` object maintains the memory of the return value; this return value must not be freed by the caller.

This method is valid to use if the kind of `TypeCode` has a [TCKind](#) value of one of the following:

```
tk_abstract_interface
tk_alias
tk_enum
tk_except
tk_native
tk_objref
tk_struct
tk_union
tk_value
tk_value_box
```

### Exceptions

[BadKind](#)      The kind of `TypeCode` is not valid for this method.

## TypeCode::kind()

```
// Java
public abstract TCKind kind();
```

Returns the kind of the `TypeCode` which is an enumerated value of type [TCKind](#). You can use `kind()` on any `TypeCode` to help determine which other `TypeCode` methods can be invoked on the `TypeCode`.

### See Also

[CORBA::TCKind](#)

---

## TypeCode::length()

```
// Java
public abstract int length() throws BadKind;
```

For strings, wide strings, and sequences, `length()` returns the bound, with zero indicating an unbounded string or sequence. For arrays, `length()` returns the number of elements in the array. This method is valid to use if the kind of `TypeCode` has a [TCKind](#) value of one of the following:

```
tk_array
tk_sequence
tk_string
tk_wstring
```

### Exceptions

[BadKind](#) The kind of `TypeCode` is not valid for this method.

## TypeCode::member\_count()

```
// Java
public abstract int member_count() throws BadKind;
```

Returns the number of members in the type. This method is valid to use if the kind of `TypeCode` has a [TCKind](#) value of one of the following:

```
tk_enum
tk_except
tk_struct
tk_union
tk_value
```

### Exceptions

[BadKind](#) The kind of `TypeCode` is not valid for this method.

## TypeCode::member\_label()

```
// Java
public abstract Any member_label(int index)
    throws BadKind, org.omg.CORBA.TypeCodePackage.Bounds;
```

---

Returns the label of the union member. For the default member, the label is the zero octet. This method is valid to use only if the kind of `TypeCode` has a [TCKind](#) value of `tk_union`.

### Parameters

`index`            The index indicating which union member you want.

### Exceptions

[BadKind](#)            The kind of `TypeCode` is not valid for this method.

[Bounds](#)            The `index` parameter is greater than or equal to the number of members for the type.

### See Also

[CORBA::TypeCode::default\\_index\(\)](#)  
[CORBA::TypeCode::member\\_count\(\)](#)

## TypeCode::member\_name()

```
// Java
public abstract String member_name(int index)
    throws BadKind, org.omg.CORBA.TypeCodePackage.Bounds;
```

Returns the simple name of the member. Because names are local to a repository, the name returned from a `TypeCode` may not match the name of the member in any particular repository, and may even be an empty string.

### Parameters

`index`            The index indicating which member to use.

This method is valid to use if the kind of `TypeCode` has a [TCKind](#) value of one of the following:

```
tk_enum
tk_except
tk_struct
tk_union
tk_value
```

The `TypeCode` object maintains the memory of the return value; this return value must not be freed by the caller.

---

## Exceptions

- [BadKind](#) The kind of `TypeCode` is not valid for this method.
- [Bounds](#) The `index` parameter is greater than or equal to the number of members for the type.

**See Also** [CORBA::TypeCode::member\\_count\(\)](#)

## TypeCode::member\_type()

```
// Java
public abstract TypeCode member_type(int index)
    throws BadKind, org.omg.CORBA.TypeCodePackage.Bounds;
```

Returns a reference to the `TypeCode` of the member identified by `index`.

## Parameters

`index` The index indicating which member you want.

This method is valid to use if the kind of `TypeCode` has a [TCKind](#) value of one of the following:

```
tk_except
tk_struct
tk_union
tk_value
```

## Exceptions

- [BadKind](#) The kind of `TypeCode` is not valid for this method.
- [Bounds](#) The `index` parameter is greater than or equal to the number of members for the type.

**See Also** [CORBA::TypeCode::member\\_count\(\)](#)

## TypeCode::member\_visibility()

```
// Java
public short member_visibility(int index) throws BadKind, Bounds {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

---

Returns the visibility of a value type member. This method is valid to use only if the kind of `TypeCode` has a [TCKind](#) value of `tk_value`.

### Parameters

`index`            The index indicating which value type member you want.

### Exceptions

[BadKind](#)            The kind of `TypeCode` is not valid for this method.

[Bounds](#)            The `index` parameter is greater than or equal to the number of members for the type.

### See Also

[CORBA::TypeCode::member\\_count\(\)](#)

## TypeCode::name()

```
// Java
public abstract String name() throws BadKind;
```

Returns the simple name identifying the type within its enclosing scope. Because names are local to a repository, the name returned from a `TypeCode` may not match the name of the type in any particular repository, and may even be an empty string.

The `TypeCode` object maintains the memory of the return value; this return value must not be freed by the caller.

This method is valid to use if the kind of `TypeCode` has a [TCKind](#) value of one of the following:

```
tk_abstract_interface
tk_alias
tk_enum
tk_except
tk_native
tk_objref
tk_struct
tk_union
tk_value
tk_value_box
```

### Exceptions

[BadKind](#)            The kind of `TypeCode` is not valid for this method.



---

## TypeCode::type\_modifier()

```
// Java
public short type_modifier() throws BadKind {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

Returns the value modifier that applies to the value type represented by this `TypeCode`. This method is valid to use only if the kind of `TypeCode` has a [TCKind](#) value of `tk_value`.

### Exceptions

[BadKind](#)      The kind of `TypeCode` is not valid for this method.



# CORBA::TypedefDef Interface

The abstract interface `TypedefDef` is simply a base interface for interface repository interfaces that define named types. Named types are types for which a name must appear in their definition such as structures, unions, and so on. Interfaces that inherit from `TypedefDef` include:

- [AliasDef](#)
- [EnumDef](#)
- [NativeDef](#)
- [StructDef](#)
- [UnionDef](#)
- [ValueBoxDef](#)

Anonymous types such as [PrimitiveDef](#), [StringDef](#), [SequenceDef](#) and [ArrayDef](#) do not inherit from [TypedefDef](#).

```
//IDL in module CORBA.  
interface TypedefDef : Contained, IDLType {};
```

The inherited operation [describe\(\)](#) is described here.

## TypedefDef::describe()

```
//IDL  
Description describe();
```

Inherited from [Contained](#), `describe()` returns a structure of type [Contained::Description](#).

The [DefinitionKind](#) type 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](#).

### See Also

[CORBA::Contained::describe\(\)](#)  
[CORBA::Contained::Description](#)  
[CORBA::TypeDescription](#)



# CORBA::UnionDef Interface

Interface `UnionDef` represents an IDL union in the interface repository.

```
// IDL in module CORBA.  
interface UnionDef : TypedefDef {  
    readonly attribute TypeCode discriminator\_type;  
    attribute IDLType discriminator\_type\_def;  
    attribute UnionMemberSeq members;  
};
```

The inherited operation [describe\(\)](#) is also described.

## See Also

[CORBA::Contained](#)  
[CORBA::TypedefDef](#)  
[CORBA::Container::create\\_union\(\)](#)

## UnionDef::describe()

```
// IDL  
Description describe();
```

Inherited from [Contained](#) (which [TypedefDef](#) inherits), `describe()` returns a structure of type [Contained::Description](#).

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](#).

## See Also

[CORBA::TypedefDef::describe\(\)](#)

## UnionDef::discriminator\_type Attribute

```
// IDL  
readonly attribute TypeCode discriminator\_type;  
  
// Java  
org.omg.CORBA.TypeCode discriminator\_type();
```

---

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.

See Also

[CORBA::TypeCode](#)

## UnionDef::discriminator\_type\_def Attribute

```
// IDL
attribute IDLType discriminator_type_def;

// Java
org.omg.CORBA.IDLType discriminator_type_def();
void discriminator_type_def( org.omg.CORBA.IDLType _val );
```

Describes the discriminator type for this union. The attribute [discriminator\\_type](#) contains the same information.

Changing this attribute will automatically update the [discriminator\\_type](#) attribute and the [IDLType::type](#) attribute.

See Also

[CORBA::IDLType::type](#)  
[CORBA::UnionDef::discriminator\\_type](#)

## UnionDef::members Attribute

```
// Java
org.omg.CORBA.UnionMember[] members();
void members( org.omg.CORBA.UnionMember[] _val );
```

Contains a description of each union member: its name, label, and type (`type` and `type_def` contain the same information).

The `members` attribute can be modified to change the union's members. Only the name, label and `type_def` fields of each [UnionMember](#) should be set (the `type` field should be set to `_tc_void`, and it will be set automatically to the [TypeCode](#) of the `type_def` field).

See Also

[CORBA::TypedefDef](#)

# CORBA::ValueBase Class

All value types have a conventional base type called `ValueBase`. `ValueBase` serves a similar role for value types that the [Object](#) class serves for interfaces. `ValueBase` serves as an abstract base class for all value type classes. You must implement concrete value type classes that inherit from `ValueBase`. `ValueBase` provides several pure virtual reference counting methods inherited by all value type classes.

The names of these methods begin with an underscore to keep them from clashing with your application-specific methods in derived value type classes.

**See Also** [CORBA::ValueFactory](#)

## `ValueBase::_add_ref()`

Increments the reference count of a value type instance and returns a pointer to this value type.

**See Also** [CORBA::ValueBase::\\_remove\\_ref\(\)](#)

## `ValueBase::_copy_value()`

Makes a deep copy of the value type instance and returns a pointer to the copy. The copy has no connections with the original instance and has a lifetime independent of that of the original.

Portable applications should not assume covariant return types but should use downcasting to regain the most derived type of a copied value type. A covariant return type means that a class derived from `ValueBase` can override `_copy_value()` to return a pointer to the derived class rather than the base class, `ValueBase*`.

**See Also** [CORBA::ValueBase::\\_downcast\(\)](#)

---

## ValueBase::\_downcast()

Returns a pointer to the base type for a derived value type class.

### Parameters

`vt` Pointer to the value type class to be downcast.

## ValueBase::\_refcount\_value()

Returns the current value of the reference count for this value type instance.

### See Also

[CORBA::ValueBase::\\_add\\_ref\(\)](#)  
[CORBA::ValueBase::\\_remove\\_ref\(\)](#)

## ValueBase::\_remove\_ref()

Decrements the reference count of a value type instance and deletes the instance when the reference count drops to zero.

If you use `delete()` to destroy instances, you must use the `new` operator to allocate all value type instances.

### See Also

[CORBA::ValueBase::\\_add\\_ref\(\)](#)

## ValueBase::~~ValueBase() Destructor

The default destructor.

The destructor is protected to prevent direct deletion of instances of classes derived from `ValueBase`.

### See Also

[CORBA::ValueBase::ValueBase\(\)](#)

## ValueBase::ValueBase() Constructors

The default constructor.

The copy constructor. Creates a new object that is a copy of `vt`.



---

The copy constructor is protected to disallow copy construction of derived value type instances except from within derived class methods.

**Parameters**

vt                      The original value type from which a copy is made.

**See Also**            [CORBA::ValueBase::~~ValueBase\(\)](#)



# CORBA::ValueBoxDef Interface

The `ValueBoxDef` interface describes an IDL value box type in the interface repository. A value box is a value type with no inheritance or operations and with a single state member. A value box is a shorthand IDL notation used to simplify the use of value types for simple containment. It behaves like an additional namespace that contains only one name.

```
// IDL in module CORBA.  
interface ValueBoxDef : IDLType {  
    attribute IDLType original\_type\_def;  
};
```

The inherited [type](#) attribute is also described.

**See Also** [CORBA::Container::create\\_value\\_box\(\)](#)

## ValueBoxDef::original\_type\_def Attribute

```
// IDL  
attribute IDLType original_type_def;  
  
// Java  
org.omg.CORBA.IDLType original_type_def();  
void original_type_def( org.omg.CORBA.IDLType _val );
```

Identifies the IDL `type_def` that is being “boxed”. Setting the `original_type_def` attribute also updates the `type` attribute.

**See Also** [CORBA::ValueBoxDef::type](#)

## ValueBoxDef::type Attribute

```
// IDL  
readonly attribute TypeCode type;
```

Inherited from [IDLType](#), this attribute is a `tk_value_box` [TypeCode](#) describing the value box.

**See Also** [CORBA::IDLType::type](#)



# CORBA::ValueDef Interface

A `ValueDef` object represents an IDL value type definition in the interface repository. It can contain constants, types, exceptions, operations, and attributes.

A `ValueDef` used as a [Container](#) may only contain [TypedefDef](#), (including definitions derived from [TypedefDef](#)), [ConstantDef](#), and [ExceptionDef](#) definitions.

```
// IDL in module CORBA.
interface ValueDef : Container, Contained, IDLType {

    // read/write interface
    attribute InterfaceDef supported\_interfaces;
    attribute InitializerSeq initializers;
    attribute ValueDef base\_value;
    attribute ValueDefSeq abstract\_base\_values;
    attribute boolean is\_abstract;
    attribute boolean is\_custom;

    // read interface
    boolean is\_a(
        in RepositoryId id
    );
    struct FullValueDescription {
        Identifier name;
        RepositoryId id;
        boolean is_abstract;
        boolean is_custom;
        RepositoryId defined_in;
        VersionSpec version;
        OpDescriptionSeq operations;
        AttrDescriptionSeq attributes;
        ValueMemberSeq members;
        InitializerSeq initializers;
        RepositoryIdSeq supported_interfaces;
        RepositoryIdSeq abstract_base_values;
        RepositoryId base_value;
        TypeCode type;
    };
};
```

---

```

};
FullValueDescription describe\_value\(\);
ValueMemberDef create\_value\_member(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType type,
    in Visibility access
);
AttributeDef create\_attribute(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType type,
    in AttributeMode mode
);
OperationDef create\_operation(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType result,
    in OperationMode mode,
    in ParDescriptionSeq params,
    in ExceptionDefSeq exceptions,
    in ContextIdSeq contexts
);
}; // End ValueDef Interface

```

The inherited [describe\(\)](#) and [contents\(\)](#) operations are also described.

**See Also** [CORBA::Container::create\\_value\(\)](#)

## ValueDef::abstract\_base\_values Attribute

```

// Java
org.omg.CORBA.ValueDef[] abstract_base_values();
void abstract_base_values( org.omg.CORBA.ValueDef[] _val );

```

The `abstract_base_values` attribute lists the abstract value types from which this value inherits.

---

## Exceptions

`BAD_PARAM,`      The `name` attribute of any object contained by this `ValueDef`  
minor code 5      conflicts with the `name` attribute of any object contained by  
any of the specified bases.

## `ValueDef::base_value` Attribute

```
// Java
org.omg.CORBA.ValueDef base_value();
void base_value( org.omg.CORBA.ValueDef _val );
```

The `base_value` attribute describes the value type from which this value inherits.

## Parameters

`BAD_PARAM,`      The `name` attribute of any object contained by the minor code  
minor code 5      5 is raised if the `name` attribute of any object contained by this  
`ValueDef` conflicts with the `name` attribute of any object con-  
tained by any of the specified bases.

## `ValueDef::contents()`

```
// IDL
ContainedSeq contents(
    in DefinitionKind limit_type,
    in boolean exclude_inherited
);
```

Inherited from [Container](#), `contents()` returns the list of constants, types, and exceptions defined in this `ValueDef` and the list of attributes, operations, and members either defined or inherited in this `ValueDef`.

---

## Parameters

<code>limit_type</code>	If set to <code>dk_all</code> , all of the contained objects in the <code>ValueDef</code> are returned. If set to the <a href="#">DefinitionKind</a> for a specific interface type, it returns only interfaces of that type. For example, if set to <code>dk_Operation</code> , then it returns contained operations only.
<code>exclude_inherited</code>	Applies only to interfaces. If true, only attributes, operations and members defined within this value type are returned. If false, all attributes, operations and members are returned.

## See Also

[CORBA::Container::contents\(\)](#)

## ValueDef::create\_attribute()

```
// Java
org.omg.CORBA.AttributeDef create_attribute(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    org.omg.CORBA.IDLType type,
    org.omg.CORBA.AttributeMode mode
);
```

Returns a new [AttributeDef](#) object contained in the `ValueDef` on which it is invoked.

## Parameters

<code>id</code>	The repository ID to use for the new <a href="#">AttributeDef</a> . An <a href="#">AttributeDef</a> inherits the <code>id</code> attribute from <a href="#">Contained</a> .
<code>name</code>	The name to use for the new <a href="#">AttributeDef</a> . An <a href="#">AttributeDef</a> inherits the <code>name</code> attribute from <a href="#">Contained</a> .
<code>version</code>	The version to use for the new <a href="#">AttributeDef</a> . An <a href="#">AttributeDef</a> inherits the <code>version</code> attribute from <a href="#">Contained</a> .
<code>type</code>	The IDL data type for the new <a href="#">AttributeDef</a> . Both the <code>type_def</code> and <code>type</code> attributes are set for <a href="#">AttributeDef</a> .
<code>mode</code>	The read or read/write <code>mode</code> to use for the new <a href="#">AttributeDef</a> .



---

The `defined_in` attribute (which the [AttributeDef](#) inherits from [Contained](#)) is initialized to identify the containing `ValueDef`.

## Exceptions

<code>BAD_PARAM,</code> minor code 5	The name attribute of any object contained by minor code 2 is raised if an object with the specified <code>id</code> already exists in the repository.
<code>BAD_PARAM,</code> minor code 3	An object with the same name already exists in this <code>ValueDef</code> .

## See Also

[CORBA::AttributeDef](#)  
[CORBA::Contained](#)

## ValueDef::create\_operation()

```
// Java
org.omg.CORBA.OperationDef create_operation(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    org.omg.CORBA.IDLType result,
    org.omg.CORBA.OperationMode mode,
    org.omg.CORBA.ParameterDescription[] params,
    org.omg.CORBA.ExceptionDef[] exceptions,
    java.lang.String[] contexts
);
```

Returns a new [OperationDef](#) object contained in the `ValueDef` on which it is invoked.

## Parameters

<code>id</code>	The repository ID to use for the new <a href="#">OperationDef</a> . An <a href="#">OperationDef</a> inherits the <code>id</code> attribute from <a href="#">Contained</a> .
<code>name</code>	The name to use for the new <a href="#">OperationDef</a> . An <a href="#">OperationDef</a> inherits the <code>name</code> attribute from <a href="#">Contained</a> .
<code>version</code>	The <code>version</code> to use for the new <a href="#">OperationDef</a> . An <a href="#">OperationDef</a> inherits the <code>version</code> attribute from <a href="#">Contained</a> .

---

result	The IDL data type of the return value for the new <a href="#">OperationDef</a> . Both the <code>result_def</code> and <code>result</code> attributes are set for the <a href="#">OperationDef</a> .
mode	The mode to use for the new <a href="#">OperationDef</a> . Specifies whether the operation is normal ( <a href="#">OP_NORMAL</a> ) or oneway ( <a href="#">OP_ONEWAY</a> ).
params	The parameters for this <a href="#">OperationDef</a> .
exceptions	The list of exceptions to use for the <a href="#">OperationDef</a> . These are exceptions the operation can raise.
contexts	The list of context identifiers to use for the <a href="#">OperationDef</a> . These represent the context clause of the operation.

The `defined_in` attribute (which the [OperationDef](#) inherits from [Contained](#)) is initialized to identify the containing `ValueDef`.

## Exceptions

BAD_PARAM, minor code 5	The <code>name</code> attribute of any object contained by minor code 2 is raised if an object with the specified <code>id</code> already exists in the repository.
BAD_PARAM, minor code 3	An object with the same <code>name</code> already exists in this <code>ValueDef</code> .

## See Also

[CORBA::OperationDef](#)  
[CORBA::Contained](#)

## ValueDef::create\_value\_member()

```
// Java
org.omg.CORBA.ValueMemberDef create_value_member(
    java.lang.String id,
    java.lang.String name,
    java.lang.String version,
    org.omg.CORBA.IDLType type,
    short access
);
```

Returns a new [ValueMemberDef](#) contained in the `valueDef` on which it is invoked.

---

## Parameters

<code>id</code>	The repository ID to use for the new <a href="#">ValueMemberDef</a> . An <a href="#">ValueMemberDef</a> inherits the <code>id</code> attribute from <a href="#">Contained</a> .
<code>name</code>	The name to use for the new <a href="#">ValueMemberDef</a> . An <a href="#">ValueMemberDef</a> inherits the <code>name</code> attribute from <a href="#">Contained</a> .
<code>version</code>	The <code>version</code> to use for the new <a href="#">ValueMemberDef</a> . An <a href="#">ValueMemberDef</a> inherits the <code>version</code> attribute from <a href="#">Contained</a> .
<code>type</code>	The IDL data type for the new <a href="#">ValueMemberDef</a> . Both the <code>type_def</code> and <code>type</code> attributes are set for <a href="#">ValueMemberDef</a> .
<code>access</code>	The visibility to use for the new <a href="#">ValueMemberDef</a> . IDL value types can have state members that are either public or private.

The `defined_in` attribute (which the [ValueMemberDef](#) inherits from [Contained](#)) is initialized to identify the containing `ValueDef`.

## Exceptions

<code>BAD_PARAM</code> , minor code 5	The <code>name</code> attribute of any object contained by minor code 2 is raised if an object with the specified <code>id</code> already exists in the repository.
<code>A BAD_PARAM</code> , minor code 3	An object with the same <code>name</code> already exists in this <code>valueDef</code> .

## See Also

[CORBA::ValueMemberDef](#)  
[CORBA::Contained](#)

## ValueDef::describe()

```
// IDL  
ValueDescription describe();
```

Inherited from [Contained](#), `describe()` for a `ValueDef` returns a [ValueDescription](#) object. Use [describe\\_value\(\)](#) for a full description of the value.

## See Also

[CORBA::ValueDescription](#)  
[CORBA::Contained::describe\(\)](#)  
[CORBA::ValueDef::describe\\_value\(\)](#)

---

## ValueDef::describe\_value()

```
// Java
org.omg.CORBA.ValueDefPackage.FullValueDescription
describe_value();
```

Returns a [FullValueDescription](#) object describing the value, including its operations and attributes.

### See Also

[CORBA::FullValueDescription](#)  
[CORBA::ValueDef::describe\(\)](#)

## ValueDefPackage.FullValueDescription.FullValueDescription()

```
// Java
public FullValueDescription(
    java.lang.String name,
    java.lang.String id,
    boolean is_abstract,
    boolean is_custom,
    byte flags,
    java.lang.String defined_in,
    java.lang.String version,
    org.omg.CORBA.OperationDescription[] operations,
    org.omg.CORBA.AttributeDescription[] attributes,
    org.omg.CORBA.ValueMember[] members,
    org.omg.CORBA.Initializer[] initializers,
    java.lang.String supported_interface,
    java.lang.String[] abstract_base_values,
    boolean has_safe_base,
    java.lang.String base_value,
    org.omg.CORBA.TypeCode type
)
```

A full description of a value type in the interface repository.

name	The name of the value type.
id	The repository ID of the value type.
is_abstract	Has a value of 1 (true) if the value is an abstract value type. A value of 0 is false.

---

<code>is_custom</code>	Has a value of 1 (true) if the value uses custom marshalling. A value of 0 is false.
<code>defined_in</code>	The repository ID that identifies where this value type is defined.
<code>version</code>	The version of the value type.
<code>operations</code>	A list of operations that the value type supports.
<code>attributes</code>	A list of attributes that the value type supports.
<code>members</code>	A list of value type members.
<code>initializers</code>	A list of initializer values for the value type.
<code>supported_interfaces</code>	A list of interfaces this value type supports.
<code>abstract_base_values</code>	A list of repository IDs that identify abstract base values.
<code>base_value</code>	A repository ID that identifies a base value.
<code>type</code>	The IDL type of the value type.

**See Also** [CORBA::ValueDef::describe\\_value\(\)](#)

### ValueDef::initializers Attribute

```
// Java
org.omg.CORBA.Initializer[] initializers();
void initializers( org.omg.CORBA.Initializer[] _val );
```

Lists the initializers this value type supports.

### ValueDef::is\_a()

```
// Java
boolean is_a( java.lang.String value_id );
```

Returns 1 (true) if this value type is either identical to or inherits, directly or indirectly, from the interface or value identified by the `id` parameter. Otherwise it returns 0 (false).

---

## Parameters

`id` The repository ID of the value type or interface to compare with this value type.

### ValueDef::is\_abstract Attribute

```
// Java
boolean is_abstract();
void is_abstract( boolean _val );
```

Returns 1 (true) if this value type is an abstract value type. Otherwise it returns 0 (false).

### ValueDef::is\_custom Attribute

```
// Java
boolean is_custom();
void is_custom( boolean _val );
```

Returns 1 (true) if this value type uses custom marshalling. Otherwise it returns 0 (false).

### ValueDef::supported\_interfaces Attribute

```
// IDL
attribute InterfaceDef supported_interfaces;
```

Lists the interfaces that this value type supports.

```
// Java
org.omg.CORBA.InterfaceDef supported_interface();
void supported_interface( org.omg.CORBA.InterfaceDef _val );
```

## Exceptions

`BAD_PARAM,`  
minor code 5 The `name` attribute of any object contained by the minor code 5 is raised if the `name` attribute of any object contained by this `ValueDef` conflicts with the `name` attribute of any object contained by any of the specified bases.

# CORBA::ValueFactory

This describes the mapping of the IDL native type `CORBA::ValueFactory`. For native IDL types, each language mapping specifies how repository IDs are used to find the appropriate factory for an instance of a value type so that it may be created as it is unmarshaled off the wire.

```
// IDL in module CORBA
native ValueFactory;
```

Recall that value types allow objects to be passed by value which implies that the ORB must be able to create instances of your value type classes during unmarshaling. However, because the ORB cannot know about all potential value type classes, you must implement factory classes for those types and register them with the ORB so the ORB can create value instances when necessary.

If the ORB is unable to locate and use the appropriate factory, then a `MARSHAL` exception with a minor code is raised.

## CORBA::ValueFactory Type

[CORBA::ORB::lookup\\_value\\_factory\(\)](#)  
[CORBA::ORB::register\\_value\\_factory\(\)](#)  
[CORBA::ORB::unregister\\_value\\_factory\(\)](#)





# CORBA::ValueMemberDef Interface

The `ValueMemberDef` interface provides the definition of a value type member in the interface repository.

```
// IDL in module CORBA.  
interface ValueMemberDef : Contained {  
    readonly attribute TypeCode type;  
    attribute IDLType type\_def;  
    attribute Visibility access;  
};
```

## ValueMemberDef::access Attribute

```
// Java  
short access();  
void access( short _val );
```

Contains an indicator of the visibility of an IDL value type state member. IDL value types can have state members that are either public or private.

## ValueMemberDef::type Attribute

```
// Java  
org.omg.CORBA.TypeCode type();
```

Describes the type of this `ValueMemberDef`.

**See Also**

[CORBA::ValueMemberDef::type\\_def](#)

## ValueMemberDef::type\_def Attribute

```
// Java  
org.omg.CORBA.IDLType type_def();  
void type_def( org.omg.CORBA.IDLType _val );
```

---

Identifies the object that defines the IDL type of this `ValueMemberDef`. The same information is contained in the `type` attribute.

You can change the type of a `ValueMemberDef` by changing its `type_def` attribute. This also changes its `type` attribute.

**See Also**

[CORBA::ValueMemberDef::type](#)

# CORBA::WString\_var Class

The class `wString_var` implements the `_var` type for IDL wide strings required by the standard C++ mapping. The `wString_var` class contains a `char*` value and ensures that this is properly freed when a `wString_var` object is deallocated, for example when execution goes out of scope.

## **WString\_var::char\*()**

Converts a `wString_var` object to a `char*`.

**See Also**

[CORBA::WString\\_var::operator=\(\)](#)

## **WString\_var::in()**

Returns the proper string for use as an input parameter.

**See Also**

[CORBA::WString\\_var::out\(\)](#)

[CORBA::WString\\_var::inout\(\)](#)

[CORBA::WString\\_var::\\_retn\(\)](#)

## **WString\_var::inout()**

Returns the proper string for use as an inout parameter.

**See Also**

[CORBA::WString\\_var::in\(\)](#)

[CORBA::WString\\_var::out\(\)](#)

[CORBA::WString\\_var::\\_retn\(\)](#)

## **WString\_var::operator=() Assignment Operators**

Assignment operators allow you to assign values to a `wString_var` from a `char*` or from another `wString_var` type.

### Parameters

- `p` A character string to assign to the `wString_var`.  
`s` A `wString_var` to assign to the `wString_var`.

**See Also** [CORBA::WString\\_var::char\\*\(\)](#)

### **WString\_var::operator[]() Subscript Operators**

Return the character at the given location of the string. Subscript operators allow access to the individual characters in the string.

### Parameters

- `index` The index location in the string.

### **WString\_var::out()**

Returns the proper string for use as an output parameter.

**See Also** [CORBA::WString\\_var::in\(\)](#)  
[CORBA::WString\\_var::inout\(\)](#)  
[CORBA::WString\\_var::\\_retn\(\)](#)

### **WString\_var::WString\_var() Constructors**

The default constructor.

Constructors that convert from a `char*` to a `wString_var`.

The copy constructor.

### Parameters

- `p` The character string to convert to a `wString_var`. The `wString_var` assumes ownership of the parameter.  
`s` The original `wString_var` that is copied.

**See Also** [CORBA::WString\\_var::~~WString\\_var\(\)](#)

---

## WString\_var::~~WString\_var() Destructor

The destructor.

**See Also** [CORBA::WString\\_var::WString\\_var\(\)](#)

## WString\_var::\_retn()

Returns the proper string for use as a method's return value.

**See Also** [CORBA::WString\\_var::inout\(\)](#)  
[CORBA::WString\\_var::in\(\)](#)  
[CORBA::WString\\_var::out\(\)](#)



# CORBA::WstringDef Interface

Interface `wstringDef` represents a bounded IDL wide string type in the interface repository. A `wstringDef` object is anonymous, which means it is unnamed. Use [Repository::create\\_wstring\(\)](#) to obtain a new `WstringDef` object.

Unbounded strings are primitive types represented with the [PrimitiveDef](#) interface. Use [Repository::get\\_primitive\(\)](#) to obtain unbounded wide strings.

```
// IDL in module CORBA.  
interface WstringDef : IDLType {  
    attribute unsigned long bound;  
};
```

The inherited [type](#) attribute is also described.

## See Also

[CORBA::IDLType](#)  
[CORBA::Repository::create\\_wstring\(\)](#)  
[CORBA::PrimitiveDef](#)  
[CORBA::StringDef](#)

## WstringDef::bound Attribute

```
// IDL  
attribute unsigned long bound;  
  
// Java  
int bound();  
void bound( int _val );
```

Specifies the maximum number of characters in the wide string. This cannot be zero.

## WstringDef::type Attribute

```
// IDL  
readonly attribute TypeCode type;
```

---

The `type` attribute is inherited from interface [IDLType](#). This attribute is a `tk_wstring` [TypeCode](#) that describes the wide string.

**See Also**

[CORBA::IDLType::type](#)



# CosEventChannelAdmin Module

The `CosEventChannelAdmin` module specifies the interfaces and exceptions for connecting suppliers and consumers to an event channel. It also provides the methods for managing these connections.

It contains the following interfaces:

- [CosEventChannelAdmin::ProxyPushConsumer Interface](#)
- [CosEventChannelAdmin::ProxyPushSupplier Interface](#)
- [CosEventChannelAdmin::ProxyPullConsumer Interface](#)
- [CosEventChannelAdmin::ProxyPullSupplier Interface](#)
- [CosEventChannelAdmin::ConsumerAdmin Interface](#)
- [CosEventChannelAdmin::SupplierAdmin Interface](#)
- [CosEventChannelAdmin::EventChannel Interface](#)

## CosEventChannelAdmin Exceptions

### **exception AlreadyConnected {};**

An `AlreadyConnected` exception is raised when an attempt is made to connect an object to the event channel when that object is already connected to the channel.

### **exception TypeError {};**

The `TypeError` exception is raised when a proxy object tries to connect an object that does not support the proper typed interface.



# CosEventChannelAdmin:: ConsumerAdmin Interface

Once a consumer has obtained a reference to a `ConsumerAdmin` object (by calling `EventChannel::for_consumers()`), they can use this interface to obtain a proxy supplier. This is necessary in order to connect to the event channel.

```
interface ConsumerAdmin
{
    ProxyPushSupplier obtain_push_supplier();
    ProxyPullSupplier obtain_pull_supplier();
};
```

## ConsumerAdmin::obtain\_push\_supplier()

```
//IDL
ProxyPushSupplier obtain_push_supplier();
```

Returns a [ProxyPushSupplier](#) object. The consumer can then use this object to connect to the event channel as a push-style consumer.

## ConsumerAdmin::obtain\_pull\_supplier()

```
//IDL
ProxyPullSupplier obtain_pull_supplier();
```

Returns a [ProxyPullSupplier](#) object. The consumer can then use this object to connect to the event channel as a pull-style consumer.



# CosEventChannelAdmin:: EventChannel Interface

The EventChannel interface lets consumers and suppliers establish a logical connection to the event channel.

```
interface EventChannel
{
    ConsumerAdmin for_consumers();
    SupplierAdmin for_suppliers();
    void destroy();
};
```

## **EventChannel::for\_consumers()**

```
//IDL
ConsumerAdmin for_consumers();
```

Used by a consumer to obtain an object reference that supports the ConsumerAdmin interface.

## **EventChannel::for\_suppliers()**

```
//IDL
SupplierAdmin for_suppliers();
```

Used by a supplier to obtain an object reference that supports the SupplierAdmin interface.

## **EventChannel::destroy()**

```
//IDL
void destroy();
```

---

Destroys the event channel. All events that are not yet delivered, as well as all administrative objects created by the channel, are also destroyed. Connected pull consumers and push suppliers are notified when their channel is destroyed.

# CosEventChannelAdmin:: ProxyPullConsumer Interface

After a supplier has obtained a reference to a proxy consumer using the [SupplierAdmin](#) interface, they use the `ProxyPullConsumer` interface to connect to the event channel.

```
interface ProxyPullConsumer : CosEventComm::PushConsumer
{
    void connect_pull_supplier(
        in CosEventComm::PullSupplier pull_supplier)
        raises (AlreadyConnected, TypeError);
};
```

## ProxyPullConsumer::connect\_pull\_supplier()

```
//IDL
void connect_pull_supplier(
    in CosEventComm::PullSupplier pull_supplier)
raises (AlreadyConnected, TypeError);
```

This operation connects the supplier to the event channel.

If the proxy pull consumer is already connected to a [PushSupplier](#), then the `AlreadyConnected` exception is raised. The `TypeError` exception is raised when supplier that is being connected does not support the proper typed event structure.

### Parameters

`pull_supplier` The supplier that is trying to connect to the event channel.





# CosEventChannelAdmin:: ProxyPullSupplier Interface

After a consumer has obtained a proxy supplier using the [ConsumerAdmin](#) interface, they use the `ProxyPullSupplier` interface to connect to the event channel.

```
interface ProxyPullSupplier : CosEventComm::PullSupplier
{
    void connect_pull_consumer(
        in CosEventComm::PullConsumer pull_consumer)
        raises (AlreadyConnected);
};
```

## **ProxyPullSupplier::connect\_pull\_consumer()**

```
//IDL
void connect_pull_consumer(
    in CosEventComm::PullConsumer pull_consumer)
raises (AlreadyConnected);
```

This operation connects the consumer to the event channel. If the consumer passes a nil object reference, the proxy pull supplier will not notify the consumer when it is about to be disconnected.

If the proxy pull supplier is already connected to the [PullConsumer](#), then the `AlreadyConnected` exception is raised.

### **Parameters**

`pull_consumer` The consumer that is trying to connect to the event channel



# CosEventChannelAdmin:: ProxyPushConsumer Interface

After a supplier has obtained a reference to a proxy consumer using the [SupplierAdmin](#) interface, they use the `ProxyPushConsumer` interface to connect to the event channel.

```
// IDL
interface ProxyPushConsumer : CosEventComm::PushConsumer
{
    void connect_push_supplier(
        in CosEventComm::PushSupplier push_supplier)
        raises (AlreadyConnected);
};
```

## ProxyPushConsumer::connect\_push\_supplier()

```
//IDL
void connect_push_supplier(
    in CosEventComm::PushSupplier push_supplier)
raises (AlreadyConnected);
```

This operation connects the supplier to the event channel. If the supplier passes a nil object reference, the proxy push consumer will not notify the supplier when it is about to be disconnected.

If the proxy push consumer is already connected to the [PushSupplier](#), then the `AlreadyConnected` exception is raised.

### Parameters

`push_supplier` The supplier that is trying to connect to the event channel



# CosEventChannelAdmin:: ProxyPushSupplier Interface

After a consumer has obtained a proxy supplier using the [ConsumerAdmin](#) interface, they use the `ProxyPushSupplier` interface to connect to the event channel.

```
interface ProxyPushSupplier : CosEventComm::PushSupplier
{
    void connect_push_consumer(
        in CosEventComm::PushConsumer push_consumer)
    raises (AlreadyConnected, TypeError);
};
```

## **ProxyPushSupplier::connect\_push\_consumer()**

```
//IDL
void connect_push_consumer(
    in CosEventComm::PushConsumer push_consumer )
raises (AlreadyConnected, TypeError);
```

This operation connects the consumer to the event channel.

If the proxy push supplier is already connected to the [PushConsumer](#), then the `AlreadyConnected` exception is raised. The `TypeError` exception is when the consumer that is being connected does not support the proper typed event structure.

### **Parameters**

`push_consumer` The consumer that is trying to connect to the event channel



# CosEventChannelAdmin:: SupplierAdmin Interface

Once a supplier has obtained a reference to a `SupplierAdmin` object (by calling `EventChannel::for_suppliers()`), they can use this interface to obtain a proxy consumer. This is necessary in order to connect to the event channel.

```
interface SupplierAdmin
{
    ProxyPushConsumer obtain_push_consumer();
    ProxyPullConsumer obtain_pull_consumer();
};
```

## **SupplierAdmin::obtain\_push\_consumer()**

```
//IDL
ProxyPushConsumer obtain_push_consumer();
```

Returns a `ProxyPushConsumer` object. The supplier can then use this object to connect to the event channel as a push-style supplier.

## **SupplierAdmin::obtain\_pull\_consumer()**

```
//IDL
ProxyPullConsumer obtain_pull_consumer();
```

Returns a `ProxyPullConsumer` object. The supplier can then use this object to connect to the event channel as a pull-style supplier.

---



# CosEventComm Module

The `CosEventComm` module specifies the interfaces which define the event service consumers and suppliers.

## CosEventComm Exceptions

### **CosEventComm::Disconnected**

```
exception Disconnected {};
```

`Disconnected` is raised when an attempt is made to contact a proxy that has not been connected to an event channel.



# CosEventComm::PullConsumer Interface

A pull-style consumer supports the `PullConsumer` interface.

```
interface PullConsumer
{
    void disconnect_pull_consumer();
};
```

## **PullConsumer::disconnect\_pull\_consumer()**

```
//IDL
void disconnect_pull_consumer();
```

Lets the supplier terminate event communication. This operation releases resources used at the consumer to support the event communication. The `PullConsumer` object reference is discarded.



# CosEventComm::PullSupplier Interface

A pull-style supplier supports the `PullSupplier` interface to transmit event data. A consumer requests event data from the supplier by invoking either the `pull()` operation or the `try_pull()` operation.

```
interface PullSupplier
{
    any pull() raises (Disconnected);
    any try_pull(out boolean has_event) raises (Disconnected);
    void disconnect_pull_supplier();
};
```

## **PullSupplier::pull()**

```
//IDL
any pull() raises (Disconnected);
```

The consumer requests event data by calling this operation. The operation blocks until the event data is available, in which case it returns the event data to the consumer. Otherwise an exception is raised. If the event communication has already been disconnected, the `OBJECT_NOT_EXIST` exception is raised.

## **PullSupplier::try\_pull()**

```
//IDL
any try_pull(out boolean has_event) raises (Disconnected);
```

Unlike the `try` operation, this operation does not block. If the event data is available, it returns the event data and sets the `has_event` parameter to true. If the event is not available, it sets the `has_event` parameter to false and the event data is returned with an undefined value. If the event communication has already been disconnected, the `OBJECT_NOT_EXIST` exception is raised.

---

## Parameters

`has_event` Indicates whether event data is available to the `try_pull` operation

## **PullSupplier::disconnect\_pull\_supplier()**

```
//IDL  
void disconnect_pull_supplier();
```

Lets the consumer terminate event communication. This operation releases resources used at the supplier to support the event communication. The `PullSupplier` object reference is discarded.

# CosEventComm::PushConsumer Interface

A push-style consumer supports the `PushConsumer` interface to receive event data.

```
interface PushConsumer
{
    void push(in any data) raises(Disconnected);
    void disconnect_push_consumer();
};
```

## PushConsumer::push()

```
//IDL
void push(in any data) raises(Disconnected);
```

Used by a supplier to communicate event data to the consumer. The supplier passes the event data as a parameter of type `any`. If the event communication has already been disconnected, the `OBJECT_NOT_EXIST` exception is raised.

### Parameters

`data`            The event data, of type `any`.

## PushConsumer::disconnect\_push\_consumer()

```
//IDL
void disconnect_push_consumer();
```

Lets the supplier terminate event communication. This operation releases resources used at the consumer to support the event communication. The `PushConsumer` object reference is discarded.





# CosEventComm::PushSupplier Interface

A push-style supplier supports the `PushSupplier` interface.

```
interface PushSupplier
{
    void disconnect_push_supplier();
};
```

## **PushSupplier::disconnect\_push\_supplier()**

```
//IDL
void disconnect_push_supplier();
```

Lets the consumer terminate event communication. This operation releases resources used at the supplier to support the event communication. The `PushSupplier` object reference is discarded.



# CosNaming Overview

The `CosNaming` module contains all IDL definitions for the CORBA naming service. The interfaces consist of:

- “[CosNaming::BindingIterator Interface](#)”
- “[CosNaming::NamingContext Interface](#)”
- “[CosNaming::NamingContextExt Interface](#)”

Use the [NamingContext](#) and [BindingIterator](#) interfaces to access standard naming service functionality. Use the [NamingContextExt](#) interface to use URLs and string representations of names.

The rest of this chapter describes data types common to the `CosNaming` module that are defined directly within its scope.

## CosNaming::Binding Structure

```
// IDL
struct Binding {
    Name binding_name;
    BindingType binding_type;
};
```

A `Binding` structure represents a single binding in a naming context. A `Binding` structure indicates the name and type of the binding:

<code>binding_name</code>	The full compound name of the binding.
<code>binding_type</code>	The binding type, indicating whether the name is bound to an application object or a naming context.

When browsing a naming graph in the naming service, an application can list the contents of a given naming context, and determine the name and type of each binding in it. To do this, the application calls the [NamingContext.list\(\)](#) method on the target [NamingContext](#) object. This method returns a list of [Binding](#) structures.

### See Also

[CosNaming::BindingList](#)  
[CosNaming::BindingType](#)

---

[NamingContext::list\(\)](#)

## CosNaming::BindingList Sequence

```
// IDL
typedef sequence<Binding> BindingList;
```

A sequence containing a set of [Binding](#) structures, each of which represents a single name binding.

An application can list the bindings in a given naming context using the [NamingContext::list\(\)](#) method. An output parameter of this method returns a value of type `BindingList`.

### See Also

[CosNaming::Binding](#)  
[CosNaming::BindingType](#)  
[NamingContext::list\(\)](#)

[“About Sequences”](#)

## CosNaming::BindingType Enumeration

```
// IDL
enum BindingType {nobject, ncontext};
```

The enumerated type `BindingType` represents these two forms of name bindings:

<code>nobject</code>	Describes a name bound to an application object.
<code>ncontext</code>	Describes a name bound to a naming context in the naming service.

There are two types of name binding in the CORBA naming service: names bound to application objects, and names bound to naming contexts. Names bound to application objects cannot be used in a compound name, except as the last element in that name. Names bound to naming contexts can be used as any component of a compound name and allow you to construct a naming graph in the naming service.

Name bindings created using [NamingContext::bind\(\)](#) or [NamingContext::rebind\(\)](#) are `nobject` bindings.

---

Name bindings created using the operations [NamingContext::bind\\_context\(\)](#) or [NamingContext::rebind\\_context\(\)](#) are ncontext bindings.

**See Also**

[CosNaming::Binding](#)  
[CosNaming::BindingList](#)

## CosNaming::Istring Data Type

```
// IDL
typedef string Istring;
```

Type `Istring` is a place holder for an internationalized string format.

## CosNaming::Name Sequence

```
// IDL
typedef sequence<NameComponent> Name;
```

A `Name` represents the name of an object in the naming service. If the object name is defined within the scope of one or more naming contexts, the name is a compound name. For this reason, type `Name` is defined as a sequence of name components.

Two names that differ only in the contents of the `kind` field of one [NameComponent](#) structure are considered to be different names.

Names with no components, that is sequences of length zero, are illegal.

**See Also**

[CosNaming::NameComponent](#)  
“About Sequences”

## CosNaming::NameComponent Structure

```
// IDL
struct NameComponent {
    Istring id;
    Istring kind;
};
```

---

A `NameComponent` structure represents a single component of a name that is associated with an object in the naming service. The members consist of:

<code>id</code>	The identifier that corresponds to the name of the component.
<code>kind</code>	The element that adds secondary type information to the component name.

The `id` field is intended for use purely as an identifier. The semantics of the `kind` field are application-specific and the naming service makes no attempt to interpret this value.

A name component is uniquely identified by the combination of both `id` and `kind` fields. Two name components that differ only in the contents of the `kind` field are considered to be different components.

**See Also**

[CosNaming::Name](#)

# CosNaming::BindingIterator Interface

A `CosNaming.BindingIterator` object stores a list of name bindings and allows application to access the elements of this list.

The [NamingContext.list\(\)](#) method obtains a list of bindings in a naming context. This method allows applications to specify a maximum number of bindings to be returned. To provide access to all other bindings in the naming context, the method returns an object of type `CosNaming.BindingIterator`.

```
// IDL
// In module CosNaming
interface BindingIterator {
    boolean next\_one(
        out Binding b
    );
    boolean next\_n(
        in unsigned long how_many,
        out BindingList bl
    );
    void destroy();
};
```

**See Also**

[CosNaming::NamingContext::list\(\)](#)

## BindingIterator::destroy()

```
// IDL
void destroy();
```

Deletes the `CosNaming::BindingIterator` object on which it is called.

## BindingIterator::next\_n()

```
// IDL
boolean next_n(
    in unsigned long how_many,
    out BindingList bl
```

---

```
);
```

Gets the next `how_many` elements in the list of bindings, subsequent to the last element obtained by a call to `next_n()` or [next\\_one\(\)](#). If the number of elements in the list is less than the value of `how_many`, all the remaining elements are obtained.

Returns `true` if one or more bindings are obtained, but returns `false` if no more bindings remain.

### Parameters

`how_many`      The maximum number of bindings to be obtained in parameter `bl`.

`bl`              The list of name bindings.

### See Also

[CosNaming::BindingIterator::next\\_one\(\)](#)  
[CosNaming::BindingList](#)

## BindingIterator::next\_one()

```
// IDL
boolean next_one(
    out Binding b
);
```

Gets the next element in the list of bindings, subsequent to the last element obtained by a call to [next\\_n\(\)](#) or `next_one()`.

Returns `true` if a binding is obtained, but returns `false` if no more bindings remain.

### Parameters

`b`                The name binding.

### See Also

[CosNaming::BindingIterator::next\\_n\(\)](#)  
[CosNaming::Binding](#)



# CosNaming::NamingContext Interface

The interface `CosNaming::NamingContext` provides operations to access the main features of the CORBA naming service, such as binding and resolving names. Name bindings are the associations the naming service maintains between an object reference and a useful name for that reference.

```
// IDL
// In module CosNaming
interface NamingContext {
    enum NotFoundReason {missing_node, not_context, not_object};

    exception NotFound {
        NotFoundReason why;
        Name rest_of_name;
    };
    exception CannotProceed {
        NamingContext cxt;
        Name rest_of_name;
    };
    exception InvalidName {};
    exception AlreadyBound {};
    exception NotEmpty {};

    void bind(
        in Name n,
        in Object obj
    )
        raises (NotFound, CannotProceed, InvalidName,
AlreadyBound);

    void rebind(
        in Name n,
        in Object obj
    )
        raises (NotFound, CannotProceed, InvalidName );

    void bind\_context(
        in Name n,
```

---

```
        in NamingContext nc
    )
    raises (NotFound, CannotProceed, InvalidName,
AlreadyBound);

void rebind\_context(
    in Name n,
    in NamingContext nc
)
    raises (NotFound, CannotProceed, InvalidName );

Object resolve(
    in Name n
)
    raises (NotFound, CannotProceed, InvalidName );

void unbind(
    in Name n
)
    raises (NotFound, CannotProceed, InvalidName );

NamingContext new\_context();

NamingContext bind\_new\_context(
    in Name n
)
    raises (NotFound, CannotProceed, InvalidName,
AlreadyBound);

void destroy() raises (NotEmpty);

void list(
    in unsigned long how_many,
    out BindingList bl,
    out BindingIterator bi
);
};
```

---

## NamingContext::AlreadyBound Exception

```
// IDL
exception AlreadyBound {};
```

If an application calls a method that attempts to bind a name to an object or naming context, but the specified name has already been bound, the method throws an exception of type `AlreadyBound`.

The following methods can throw this exception:

```
bind\(\)
bind\_context\(\)
bind\_new\_context\(\)
```

## NamingContext::bind()

```
// IDL
void bind(
    in Name n,
    in Object obj
)
    raises (NotFound, CannotProceed, InvalidName, AlreadyBound);
```

Creates a name binding, relative to the target naming context, between a name and an object.

### Parameters

<code>n</code>	The name to be bound to the target object, relative to the naming context on which the method is called.
<code>obj</code>	The application object to be associated with the specified name.

If the name passed to this method is a compound name with more than one component, all except the last component are used to find the sub-context in which to add the name binding.

### Exceptions

The method can throw these exceptions:

```
NotFound
CannotProceed
InvalidName
AlreadyBound
```

---

The contexts associated with the components must already exist, otherwise the method throws a [NotFound](#) exception.

### See Also

[CosNaming::NamingContext::rebind\(\)](#)  
[CosNaming::NamingContext::resolve\(\)](#)

## NamingContext::bind\_context()

```
// IDL
void bind_context(
    in Name n,
    in NamingContext nc
)
    raises (NotFound, CannotProceed, InvalidName, AlreadyBound);
```

Creates a binding, relative to the target naming context, between a name and another, specified naming context.

### Parameters

- n** The name to be bound to the target naming context, relative to the naming context on which the method is called. All but the final naming context specified in parameter **n** must already exist.
- nc** The [NamingContext](#) object to be associated with the specified name. This object must already exist. To create a new [NamingContext](#) object, call [NamingContext::new\\_context\(\)](#). The entries in naming context **nc** can be resolved using compound names.

This new binding can be used in any subsequent name resolutions. The naming graph built using `bind_context()` is not restricted to being a tree: it can be a general naming graph in which any naming context can appear in any other naming context.

### Exceptions

The method can throw these exceptions:

[NotFound](#)  
[CannotProceed](#)  
[InvalidName](#)  
[AlreadyBound](#)

---

This method throws an [AlreadyBound](#) exception if the name specified by `n` is already in use.

### See Also

[CosNaming.NamingContext.bind\\_new\\_context\(\)](#)  
[CosNaming.NamingContext.new\\_context\(\)](#)  
[CosNaming.NamingContext.rebind\\_context\(\)](#)  
[CosNaming.NamingContext.resolve\(\)](#)

## NamingContext::bind\_new\_context()

```
// IDL
NamingContext bind_new_context(
    in Name n
)
    raises (NotFound, CannotProceed, InvalidName, AlreadyBound);
```

Creates a new [NamingContext](#) object in the naming service and binds the specified name to it, relative to the naming context on which the method is called. The method returns a reference to the newly created [NamingContext](#) object.

### Parameters

`n` The name to be bound to the newly created naming context, relative to the naming context on which the method is called. All but the final naming context specified in parameter `n` must already exist.

This method has the same effect as a call to [NamingContext::new\\_context\(\)](#) followed by a call to [NamingContext::bind\\_context\(\)](#).

The new name binding created by this method can be used in any subsequent name resolutions: the entries in the returned naming context can be resolved using compound names.

### Exceptions

The method can throw these exceptions:

[NotFound](#)  
[CannotProceed](#)  
[InvalidName](#)  
[AlreadyBound](#)

This method throws an [AlreadyBound](#) exception if the name specified by `n` is already in use.

---

**See Also**

[CosNaming::NamingContext::bind\\_context\(\)](#)  
[CosNaming::NamingContext::new\\_context\(\)](#)

## NamingContext::CannotProceed Exception

```
// IDL
exception CannotProceed {
    NamingContext cxt;
    Name rest_of_name;
};
```

If a naming service method fails due to an internal error, the method throws a CannotProceed exception.

A CannotProceed exception consists of two member fields:

cxt	The <a href="#">NamingContext</a> object associated with the component at which the method failed.
rest_of_name	The remainder of the compound name, after the binding for the component at which the method failed.

The application might be able to use the information returned in this exception to complete the method later. For example, if you use a naming service federated across several hosts and one of these hosts is currently unavailable, a naming service method might fail until that host is available again.

The following methods can throw this exception:

[bind\(\)](#)  
[bind\\_context\(\)](#)  
[bind\\_new\\_context\(\)](#)  
[rebind\(\)](#)  
[rebind\\_context\(\)](#)  
[resolve\(\)](#)  
[unbind\(\)](#)

**See Also**

[CosNaming::Name](#)  
[CosNaming::NamingContext](#)

---

## NamingContext::destroy()

```
// IDL
void destroy()
    raises (NotEmpty);
```

Deletes the [NamingContext](#) object on which it is called. Before deleting a [NamingContext](#) in this way, ensure that it contains no bindings.

To avoid leaving name bindings with no associated objects in the naming service, call [NamingContext.unbind\(\)](#) to unbind the context name before calling `destroy()`. See [resolve\(\)](#) for information about the result of resolving names of context objects that no longer exist.

### Exceptions

[NamingContext](#): `destroy()` is called on a [NamingContext](#) that contains existing bindings.

### See Also

[CosNaming::NamingContext::resolve\(\)](#)  
[CosNaming::NamingContext::unbind\(\)](#)

## NamingContext::InvalidName Exception

```
// IDL
exception InvalidName {};
```

If a method receives an `in` parameter of type [CosNaming.Name](#) for which the sequence length is zero, the method throws an `InvalidName` exception.

The following methods can throw this exception:

```
bind\(\)  
bind\_context\(\)  
bind\_new\_context\(\)  
rebind\(\)  
rebind\_context\(\)  
resolve\(\)  
unbind\(\)
```

---

## NamingContext::list()

```
// IDL
void list(
    in unsigned long how_many,
    out BindingList bl,
    out BindingIterator bi
);
```

Gets a list of the name bindings in the naming context on which the method is called.

### Parameters

how_many	The maximum number of bindings to be obtained in the <a href="#">BindingList</a> parameter, bl.
bl	The list of bindings contained in the naming context on which the method is called.
bi	A <a href="#">BindingIterator</a> object that provides access to all remaining bindings contained in the naming context on which the method is called.  If the naming context contains more than the requested number of bindings, the <a href="#">BindingIterator</a> contains the remaining bindings. If the naming context does not contain any additional bindings, the parameter bi is a nil object reference.

### See Also

[CosNaming::BindingIterator](#)  
[CosNaming::BindingList](#)

## NamingContext::new\_context()

```
// IDL
NamingContext new_context();
```

Creates a new [NamingContext](#) object in the naming service, without binding a name to it. The method returns a reference to the newly created [NamingContext](#) object.

After creating a naming context with this method, your application can bind a name to it by calling [NamingContext::bind\\_context\(\)](#). There is no relationship between this object and the [NamingContext](#) object on which the application call the method.



---

**See Also**

[CosNaming::NamingContext::bind\\_context\(\)](#)  
[CosNaming::NamingContext::bind\\_new\\_context\(\)](#)

## NamingContext::NotEmpty Exception

```
// IDL
exception NotEmpty {};
```

An application can call the [NamingContext::destroy\(\)](#) method to delete a naming context object in the naming service. For this method to succeed, the naming context must contain no bindings. If bindings exist in the naming context, the method throws a `NotEmpty` exception.

## NamingContext::NotFound Exception

```
// IDL
exception NotFound {
    NotFoundReason why;
    Name rest_of_name;
};
```

Several methods in the interface [CosNaming::NamingContext](#) require an existing name binding to be passed as an input parameter. If such a method receives a name binding that it determines is invalid, the method throws a `NotFound` exception. This exception contains two member fields:

<code>why</code>	The reason why the name binding is invalid.
<code>rest_of_name</code>	The remainder of the compound name following the invalid portion of the name that the method determined to be invalid.

The following methods can throw this exception:

[bind\(\)](#)  
[bind\\_context\(\)](#)  
[bind\\_new\\_context\(\)](#)  
[rebind\(\)](#)  
[rebind\\_context\(\)](#)  
[resolve\(\)](#)  
[unbind\(\)](#)

---

**See Also** [CosNaming::NamingContext::NotFoundReason](#)

## NamingContext::NotFoundReason Enumeration

```
// IDL
enum NotFoundReason {missing_node, not_context, not_object};
```

If an method throws a [NotFound](#) exception, a value of enumerated type `NotFoundReason` indicates the reason why the exception was thrown. The reasons consists of:

<code>missing_node</code>	The component of the name passed to the method did not exist in the naming service.
<code>not_context</code>	The method expected to receive a name that is bound to a naming context, for example using <a href="#">NamingContext::bind_context()</a> , but the name received did not satisfy this requirement.
<code>not_object</code>	The method expected to receive a name that is bound to an application object, for example using <a href="#">NamingContext::bind()</a> , but the name received did not satisfy this requirement.

**See Also** [CosNaming::NamingContext::NotFound](#)

## NamingContext::rebind()

```
// IDL
void rebind(
    in Name n,
    in Object obj
)
    raises (NotFound, CannotProceed, InvalidName);
```

Creates a binding between an object and a name that is already bound in the target naming context. The previous name is unbound and the new binding is created in its place.

---

## Parameters

n	The name to be bound to the specified object, relative to the naming context on which the method is called.
obj	The application object to be associated with the specified name.

As is the case with [NamingContext::bind\(\)](#), all but the last component of a compound name must exist, relative to the naming context on which you call the method.

## Exceptions

The method can throw these exceptions:

[NotFound](#)  
[CannotProceed](#)  
[InvalidName](#)

## See Also

[CosNaming::NamingContext::bind\(\)](#)  
[CosNaming::NamingContext::resolve\(\)](#)

## NamingContext::rebind\_context()

```
// IDL
void rebind_context(
    in Name n,
    in NamingContext nc
)
    raises (NotFound, CannotProceed, InvalidName);
```

The `rebind_context()` method creates a binding between a naming context and a name that is already bound in the context on which the method is called. The previous name is unbound and the new binding is made in its place.

## Parameters

n	The name to be bound to the specified naming context, relative to the naming context on which the method is called.
nc	The naming context to be associated with the specified name.

As is the case for [NamingContext::bind\\_context\(\)](#), all but the last component of a compound name must name an existing [NamingContext](#).

## Exceptions

The method can throw these exceptions:

---

[NotFound](#)  
[CannotProceed](#)  
[InvalidName](#)

**See Also**

[CosNaming::NamingContext::bind\\_context\(\)](#)  
[CosNaming::NamingContext::resolve\(\)](#)

## NamingContext::resolve()

```
// IDL
Object resolve(
    in Name n
)
    raises (NotFound, CannotProceed, InvalidName);
```

Returns the object reference that is bound to the specified name, relative to the naming context on which the method was called. The first component of the specified name is resolved in the target naming context.

**Parameters**

`n`                    The name to be resolved, relative to the naming context on which the method is called.

**Exceptions**

The method can throw these exceptions:

[NotFound](#)  
[CannotProceed](#)  
[InvalidName](#)

If the name `n` refers to a naming context, it is possible that the corresponding [NamingContext](#) object no longer exists in the naming service. For example, this could happen if you call [NamingContext::destroy\(\)](#) to destroy a context without first unbinding the context name. In this case, `resolve()` throws a CORBA system exception.

**See Also**

[CosNaming::NamingContext::CannotProceed](#)  
[CosNaming::NamingContext::InvalidName](#)  
[CosNaming::NamingContext::NotFound](#)

---

## NamingContext::unbind()

```
// IDL
void unbind(
    in Name n
)
    raises (NotFound, CannotProceed, InvalidName);
```

Removes the binding between a specified name and the object associated with it.

### Parameters

**n**                      The name to be unbound in the naming service, relative to the naming context on which the method is called.

Unbinding a name does not delete the application object or naming context object associated with the name. For example, if you want to remove a naming context completely from the naming service, you should first unbind the corresponding name, then delete the [NamingContext](#) object by calling [NamingContext::destroy\(\)](#).

### Exceptions

The method can throw these exceptions:

[NotFound](#)  
[CannotProceed](#)  
[InvalidName](#)

### See Also

[CosNaming::NamingContext::CannotProceed](#)  
[CosNaming::NamingContext::destroy\(\)](#)  
[CosNaming::NamingContext::InvalidName](#)  
[CosNaming::NamingContext::NotFound](#)



# CosNaming::NamingContextExt Interface

The NamingContextExt interface, derived from [NamingContext](#), provides the capability for applications to use strings and Uniform Resource Locator (URL) strings to access names in the naming service.

```
// IDL
// In module CosNaming
interface NamingContextExt: NamingContext {
    typedef string StringName;
    typedef string Address;
    typedef string URLString;

    StringName to\_string(
        in Name n
    )
        raises(InvalidName);

    Name to\_name(
        in StringName sn
    )
        raises(InvalidName);

    exception InvalidAddress {};

    URLString to\_url(
        in Address addr,
        in StringName sn
    )
        raises(InvalidAddress, InvalidName);

    Object resolve\_str(
        in StringName n
    )
        raises(NotFound, CannotProceed, InvalidName,
AlreadyBound);
};
```

---

## NameContextExt::Address Data Type

```
// IDL
typedef string Address;
```

A URL address component is a host name optionally followed by a port number (delimited by a colon). Examples include the following:

```
my_backup_host.555xyz.com:900
myhost.xyz.com
myhost.555xyz.com
```

## NameContextExt::InvalidAddress Exception

```
// IDL
exception InvalidAddress {};
```

The [to\\_url\(\)](#) method throws an `InvalidAddress` exception when an invalid URL address component is passed to it.

### See Also

[CosNaming::NamingContextExt::to\\_url\(\)](#)

## NameContextExt::resolve\_str()

```
// IDL
Object resolve_str(
    in StringName sn
)
    raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
```

Resolves a naming service name to the object it represents in the same manner as [NamingContext::resolve\(\)](#). This method accepts a string representation of a name as an argument instead of a [Name](#) data type.

### Parameters

`sn` String representation of a name to be resolved to an object reference.

### Exceptions

The method can throw these exceptions:

[NotFound](#)  
[CannotProceed](#)  
[InvalidName](#)



---

[AlreadyBound](#)

## **NameContextExt::StringName Data Type**

```
// IDL
typedef string StringName;
```

A string representation of an object's name in the naming service.

### **See Also**

[CosNaming::Name](#)

## **NameContextExt::to\_name()**

```
// IDL
Name to_name(
    in StringName sn
)
raises(InvalidName);
```

Returns a naming service [Name](#) given a string representation of it.

### **Parameters**

sn                      String representation of a name in the naming service to be converted to a [Name](#) data type.

### **Exceptions**

[InvalidName](#)          The string name is syntactically malformed or violates an implementation limit.

## **NameContextExt::to\_string()**

```
// IDL
StringName to_string(
    in Name n
)
raises(InvalidName);
```

Returns a string representation of a naming service [Name](#) data type.

---

## Parameters

`n` The naming service [Name](#) to be converted to a string.

## Exceptions

[InvalidName](#) [Name](#) is invalid.

## NameContextExt::to\_url()

```
// IDL
URLString to_url(
    in Address addr,
    in StringName sn
)
    raises(InvalidAddress, InvalidName);
```

Returns a fully formed URL string, given a URL address component and a string representation of a name. It adds the necessary escape sequences to create a valid [URLString](#).

## Parameters

`addr` The URL address component. An empty address means the local host.

`sn` The string representation of a naming service name. An empty string is allowed.

## Exceptions

The method can throw these exceptions:

[InvalidAddress](#)  
[InvalidName](#)

## NameContextExt::URLString Data Type

```
// IDL
typedef string URLString;
```

A valid Uniform Resource Locator (URL) string. URL strings describe the location of a resource that is accessible via the Internet.

# CosNotification Module

The `CosNotification` module defines the structured event data type, and a data type used for transmitting sequences of structured events. In addition, this module provides constant declarations for each of the standard quality of service (QoS) and administrative properties supported by the notification service. Some properties also have associated constant declarations to indicate their possible settings. Finally, administrative interfaces are defined for managing sets of QoS and administrative properties.

## CosNotification Data Types

### CosNotification::StructuredEvent Data Structure

```
//IDL
struct EventType {
    string domain_name;
    string type_name;
};

struct FixedEventHeader {
    EventType event_type;
    string event_name;
};

struct EventHeader {
    FixedEventHeader fixed_header;
    OptionalHeaderFields variable_header;
};

struct StructuredEvent {

    EventHeader header;
    FilterableEventBody filterable_data;
    any remainder_of_body;
```

---

```
}; // StructuredEvent
```

The `StructuredEvent` data structure defines the fields which make up a structured event. A detailed description of structured events is provided in the *CORBA Notification Service Guide*.

### **CosNotification::EventTypeSeq Type**

```
//IDL
struct EventType {
    string domain_name;
    string type_name;
};
typedef sequence <EventType> EventTypeSeq
```

### **CosNotification::EventBatch Type**

The `CosNotification` module defines the `EventBatch` data type as a sequence of structured events. The `CosNotifyComm` module defines interfaces supporting the transmission and receipt the `EventBatch` data type.

## **QoS and Administrative Constant Declarations**

The `CosNotification` module declares several constants related to QoS properties, and the administrative properties of event channels.

```
// IDL in CosNotification module
const string EventReliability = "EventReliability";
const short BestEffort = 0;
const short Persistent = 1;

const string ConnectionReliability = "ConnectionReliability";
// Can take on the same values as EventReliability

const string Priority = "Priority";
const short LowestPriority = -32767;
const short HighestPriority = 32767;
const short DefaultPriority = 0;
```

```
const string StartTime = "StartTime";
// StartTime takes a value of type TimeBase::UtcT

const string StopTime = "StopTime";
// StopTime takes a value of type TimeBase::UtcT

const string Timeout = "Timeout";
// Timeout takes on a value of type TimeBase::TimeT

const string OrderPolicy = "OrderPolicy";
const short AnyOrder = 0;
const short FifoOrder = 1;
const short PriorityOrder = 2;
const short DeadlineOrder = 3;

const string DiscardPolicy = "DiscardPolicy";
// DiscardPolicy takes on the same values as OrderPolicy, plus
const short LifoOrder = 4;

const string MaximumBatchSize = "MaximumBatchSize";
// MaximumBatchSize takes on a value of type long

const string PacingInterval = "PacingInterval";
/ PacingInterval takes on a value of type TimeBase::TimeT

const string StartTimeSupported = "StartTimeSupported";
// StartTimeSupported takes on a boolean value

const string StopTimeSupported = "StopTimeSupported";
// StopTimeSupported takes on a boolean value

const string MaxEventsPerConsumer = "MaxEventsPerConsumer";
// MaxEventsPerConsumer takes on a value of type long
```

## QoS and Admin Data Types

The `CosNotification` module defines several data types related to QoS properties, and the administrative properties of event channels.

---

## CosNotification::PropertyName Type

```
typedef string PropertyName;
```

PropertyName is a string holding the name of a QoS or an Admin property.

## CosNotification::PropertyValue Type

```
typedef any PropertyValue;
```

PropertyValue is an any holding the setting of QoS or Admin properties.

## CosNotification::PropertySeq Type

```
//IDL in CosNotification module
struct Property
{
    PropertyName name;
    PropertyValue value;
};
typedef sequence <Property> PropertySeq;
```

PropertySeq is a set of name-value pairs that encapsulate QoS or Admin properties and their values.

name	A string identifying the QoS or Admin property.
value	An Any containing the setting of the QoS or Admin property.

## CosNotification::QoSProperties Type

```
typedef PropertySeq QoSProperties;
```

QoSProperties is a name-value pair of [PropertySeq](#) used to specify QoS properties.

## CosNotification::AdminProperties Type

```
typedef PropertySeq AdminProperties;
```

`AdminProperties` is a name-value pair of [PropertySeq](#) used to specify Admin properties.

### CosNotification::QoSError\_code Enum

```
enum QoSError_code
{
    UNSUPPORTED_PROPERTY,
    UNAVAILABLE_PROPERTY,
    UNSUPPORTED_VALUE,
    UNAVAILABLE_VALUE,
    BAD_PROPERTY,
    BAD_TYPE,
    BAD_VALUE
};
```

`QoSError_code` specifies the error codes for [UnsupportedQoS](#) and [UnsupportedAdmin](#) exceptions. The return codes are:

<code>UNSUPPORTED_PROPERTY</code>	Orbix does not support the property for this type of object
<code>UNAVAILABLE_PROPERTY</code>	This property cannot be combined with existing QoS properties.
<code>UNSUPPORTED_VALUE</code>	The value specified for this property is invalid for the target object.
<code>UNAVAILABLE_VALUE</code>	The value specified for this property is invalid in the context of other QoS properties currently in force.
<code>BAD_PROPERTY</code>	The property name is unknown.
<code>BAD_TYPE</code>	The type supplied for the value of this property is incorrect.
<code>BAD_VALUE</code>	The value specified for this property is illegal.

### CosNotification::PropertyErrorSeq Type

```
// IDL from CosNotification module
struct PropertyRange
{
```

---

```
    PropertyValue low_val;  
    PropertyValue high_val;  
};
```

```
struct PropertyError  
{  
    QoSError\_code code;  
    PropertyName name;  
    PropertyRange available_range;  
};  
typedef sequence <PropertyError> PropertyErrorSeq;
```

A `PropertyErrorSeq` is returned when [UnsupportedQoS](#) or [UnsupportedAdmin](#) is raised. It specifies a sequence containing the reason for the exception, the property that caused it, and a range of valid settings for the property.

### **CosNotification::NamedPropertyRangeSeq Type**

```
struct NamedPropertyRange  
{  
    PropertyName name;  
    PropertyRange range;  
};  
typedef sequence <NamedPropertyRange> NamedPropertyRangeSeq;
```

Specifies a range of values for the named property.

## **QoS and Admin Exceptions**

The `CosNotification` module defines two exceptions related to QoS properties, and the administrative properties of event channels.

### **CosNotification::UnsupprtedQoS**

```
exception UnsupportedQoS { PropertyErrorSeq qos_err; };
```

Raised when setting QoS properties on notification channel objects, or when validating QoS properties. It returns with a [PropertyErrorSeq](#) specifying the



reason for the exception, which property was invalid, and a list of valid settings for the QoS property.

### **CosNotification::UnsupportedAdmin**

```
exception UnsupportedAdmin { PropertyErrorSeq admin_err; };
```

Raised when setting Admin properties on notification channels. It returns with a [PropertyErrorSeq](#) specifying the reason for the exception, which property was invalid, and a list of valid settings for the property.



# CosNotification:: AdminPropertiesAdmin Interface

```
//IDL
interface AdminPropertiesAdmin {
    AdminProperites get\_admin\(\);
    void set\_admin (in AdminProperites admin)
        raises ( UnsupportedAdmin );
};
```

The `AdminPropertiesAdmin` interface defines operations enabling clients to manage the values of administrative properties. This interface is an abstract interface which is inherited by the Event Channel interfaces defined in the [CosNotifyChannelAdmin](#) module.

## **AdminPropertiesAdmin::get\_admin()**

```
AdminProperites get\_admin\(\);
```

Returns a sequence of name-value pairs encapsulating the current administrative settings for the target channel.

## **AdminPropertiesAdmin::set\_admin()**

```
void set\_admin (in AdminProperites admin)
    raises ( UnsupportedAdmin );
```

Sets the specified administrative properties on the target object.

### **Parameters**

<code>admin</code>	A sequence of name-value pairs encapsulating administrative property settings.
--------------------	--

---

## Exceptions

[UnsupportedAdmin](#) Raised if If any of the requested settings cannot be satisfied by the target object.

# CosNotification::QoSAdmin Interface

```
//IDL
interface QoSAdmin {
    QoSProperties get\_qos\(\);
    void set\_qos ( in QoSProperties qos)
        raises ( UnsupportedQoS );
    void validate\_qos (
        in QoSProperties required_qos,
        out NamedPropertyRangeSeq available_qos )
        raises ( UnsupportedQoS );
}
```

The `QoSAdmin` interface defines operations enabling clients to manage the values of QoS properties. It also defines an operation to verify whether or not a set of requested QoS property settings can be satisfied, along with returning information about the range of possible settings for additional QoS properties. `QoSAdmin` is an abstract interface which is inherited by the proxy, admin, and event channel interfaces defined in the [CosNotifyChannelAdmin](#) module.

## QoSAdmin::get\_qos()

```
QoSProperties get\_qos\(\);
```

Returns a sequence of name-value pairs encapsulating the current quality of service settings for the target object (which could be an event channel, admin, or proxy object).

## QoSAdmin::set\_qos()

```
void set\_qos ( in QoSProperties qos)
    raises ( UnsupportedQoS );
```

Sets the specified QoS properties on the target object (which could be an event channel, admin, or proxy object).



---

## Exceptions

[UnsupportedQoS](#) Raised if any of the requested settings cannot be satisfied by the target object.





# CosNotifyChannelAdmin Module

The `CosNotifyChannelAdmin` module specifies the interfaces, exceptions, and data types for connecting suppliers and consumers to an event channel. It also provides the methods for managing these connections.

## CosNotifyChannelAdmin Data Types

`CosNotifyChannelAdmin` specifies data types that facilitate the connection of clients to an event channel. The data types specify the proxy type used by a client, the type of events a client can send or receive, and how the clients receive subscription information. Several data types identify the client and the event channel objects responsible for managing it.

### CosNotifyChannelAdmin::ProxyType Enum

```
// IDL in CosNotifyChannelAdmin
enum ProxyType
{
    PUSH_ANY,
    PULL_ANY,
    PUSH_STRUCTURED,
    PULL_STRUCTURED,
    PUSH_SEQUENCE,
    PULL_SEQUENCE,
    PUSH_TYPED,
    PULL_TYPED
}
```

Specifies the type of proxy used by a client to connect to an event channel. The type of proxy must match the type of client it connects to the channel. For example, a structured push consumer must use a `PUSH_STRUCTURED` proxy.

---

## CosNotifyChannelAdmin::ObtainInfoMode Enum

```
// IDL in CosNotifyChannelAdmin Module
enum ObtainInfoMode
{
    ALL_NOW_UPDATES_ON,
    ALL_NOW_UPDATES_OFF,
    NONE_NOW_UPDATES_ON,
    NONE_NOW_UPDATES_OFF
}
```

Specifies how the client wishes to be notified of changes in subscription/publication information. The values have the following meanings:

`ALL_NOW_UPDATES_ON` Returns the current subscription/publication information and enables automatic updates.

`ALL_NOW_UPDATES_OFF` Returns the current subscription/publication information and disables automatic updates.

`NONE_NOW_UPDATES_ON` Enables automatic updates of subscription/publication information without returning the current information.

`NONE_NOW_UPDATES_OFF` Disables automatic updates of subscription/publication information without returning the current information.

## CosNotifyChannelAdmin::ProxyID Type

```
typedef long ProxyID;
```

Specifies the ID of a proxy in an event channel.

## CosNotifyChannelAdmin::ProxyIDSeq Type

```
typedef sequence <ProxyID> ProxyIDSeq
```

Contains a list of `ProxyID` values.

## CosNotifyChannelAdmin::ClientType Enum

```
// IDL in CosNotifyChannelAdmin
```

```
enum ClientType
{
    ANY_EVENT,
    STRUCTURED_EVENT,
    SEQUENCE_EVENT
}
```

Specifies the type of messages a client handles. The values have the following meanings:

ANY_EVENT	The client sends or receives messages as an <code>Any</code> . Consumers set with <code>ANY_EVENT</code> can receive structured messages, but the consumer is responsible for decoding it.
STRUCTURED_EVENT	The client sends or receives messages as a <a href="#">CosNotification::StructuredEvent</a> .
SEQUENCE_EVENT	The client sends or receives messages as a <a href="#">CosNotification::EventBatch</a> .

### CosNotifyChannelAdmin::InterFilterGroupOperator Enum

```
// IDL in CosNotifyChannelAdmin
enum InterFilterGroupOperator
{
    AND_OP,
    OR_OP
}
```

Specifies the relationship between filters set on an admin object and the filters set on its associated filter objects. The values have the following meanings:

AND_OP	Events must pass at least one filter in both the proxy and the admin in order to be forwarded along the delivery path.
OR_OP	Events must pass at least one filter in either the proxy or the admin in order to be forwarded along the delivery path.

### CosNotifyChannelAdmin::AdminID Type

```
typedef long AdminID;
```

---

Specifies the ID of an admin object in an event channel.

### **CosNotifyChannelAdmin::AdminIDSeq**

```
typedef sequence <AdminID> AdminIDSeq;
```

Contains a list of IDs for admin objects in an event channel.

### **CosNotifyChannelAdmin::AdminLimit Type**

```
//IDL in CosNotifyChannelAdmin
struct AdminLimit
{
    CosNotification::PropertyName name;
    CosNotification::PropertyValue value;
}
```

Specifies the administration property whose limit is exceeded and the value of that property. It is returned by an [CosNotifyChannelAdmin::AdminLimitExceeded](#) exception.

name            Name of the admin property that caused the exception.

value           The current value of the property.

### **CosNotifyChannelAdmin::ChannelID Type**

```
typedef long ChannelID;
```

Specifies an event channel in the notification service.

### **CosNotifyChannelAdmin::ChannelIDSeq Type**

```
typedef sequence <ChannelID> ChannelIDSeq;
```

Contains a list of IDs for event channels in the notification service.

---

## CosNotifyChannelAdmin Exceptions

The `CosNotifyChannelAdmin` module defines exceptions to handle errors generated while managing client connections to an event channel.

### **CosNotifyChannelAdmin::ConnectionAlreadyActive Exception**

```
exception ConnectionAlreadyActive{};
```

Raised when attempting to resume an already active connection between a client and an event channel.

### **CosNotifyChannelAdmin::ConnetionAlreadyInactive Exception**

```
exception ConnectionAlreadyInactive{};
```

Raised when attempting to suspend a connection between a client and an event channel while it is suspended.

### **CosNotifyChannelAdmin::NotConnected Exception**

```
exception NotCennected{};
```

Raised when attempting to suspend or resume a connection between a client and an event channel when the client is not connected to the channel.

### **CosNotifyChannelAdmin::AdminNotFound Exception**

```
exception AdminNotFound{};
```

Raised when the specified Admin ID cannot be resolved.

### **CosNotifyChannelAdmin::ProxyNotFound Exception**

```
exception ProxyNotFound{};
```

---

Raised when the specified proxy ID cannot be resolved.

### **CosNotifyChannelAdmin::AdminLimitExceeded Exception**

```
exception AdminLimitExceeded{ AdminLimit admin_property_err };
```

Raised when an attempt to obtain a proxy and the new connection will put the event channel over the limit set by its `MaxConsumers` or `MaxSuppliers` setting.

The returned [AdminLimit](#) specifies which property caused the exception and the current setting of the property.

### **CosNotifyChannelAdmin::ChannelNotFound Exception**

```
exception ChannelNotFound{};
```

Raised when the specified channel ID cannot be resolved.

# CosNotifyChannelAdmin:: ConsumerAdmin Interface

```
//IDL
interface ConsumerAdmin :
    CosNotification::QoSAdmin,
    CosNotifyComm::NotifySubscribe,
    CosNotifyFilter::FilterAdmin,
    CosEventChannelAdmin::ConsumerAdmin
{
    readonly attribute AdminID MyID;
    readonly attribute EventChannel MyChannel;

    readonly attribute InterFilterGroupOperator MyOperator;

    attribute CosNotifyFilter::MappingFilter priority_filter;
    attribute CosNotifyFilter::MappingFilter lifetime_filter;

    readonly attribute ProxyIDSeq pull_suppliers;
    readonly attribute ProxyIDSeq push_suppliers;

    ProxySupplier get_proxy_supplier ( in ProxyID proxy_id )
        raises ( ProxyNotFound );

    ProxySupplier obtain_notification_pull_supplier (
        in ClientType ctype,
        out ProxyID proxy_id)
        raises ( AdminLimitExceeded );

    ProxySupplier obtain_notification_push_supplier (
        in ClientType ctype,
        out ProxyID proxy_id)
        raises ( AdminLimitExceeded );

    ProxySupplier obtain_txn_notification_pull_supplier (
        in ClientType ctype,
        out ProxyID proxy_id)
```

---

```
        raises ( AdminLimitExceeded );  
  
        void destroy();  
};
```

The `ConsumerAdmin` interface defines the behavior of objects that create and manage lists of proxy supplier objects within an event channel. A event channel can have any number of `ConsumerAdmin` instances associated with it. Each instance is responsible for creating and managing a list of proxy supplier objects that share a common set of QoS property settings, and a common set of filter objects. This feature enables clients to group proxy suppliers within a channel into groupings that each support a set of consumers with a common set of QoS requirements and event subscriptions.

The `ConsumerAdmin` interface inherits the [QoSAdmin](#) interface defined within [CosNotification](#), enabling each `ConsumerAdmin` to manage a set of QoS property settings. These QoS property settings are assigned as the default QoS property settings for any proxy supplier object created by a `ConsumerAdmin`. The `ConsumerAdmin` interface also inherits from the [FilterAdmin](#) interface defined within [CosNotifyFilter](#). This enables each `ConsumerAdmin` to maintain a list of filters. These filters encapsulate subscriptions that apply to all proxy supplier objects that have been created by a given `ConsumerAdmin`.

The `ConsumerAdmin` interface also inherits from the [NotifySubscribe](#) interface defined in [CosNotifyComm](#). This inheritance enables a `ConsumerAdmin` to be registered as the callback object for notification of subscription changes made on filters. This optimizes the notification of a group of proxy suppliers that have been created by the same `ConsumerAdmin` of changes to these shared filters.

The `ConsumerAdmin` interface also inherits from `CosEventChannelAdmin::ConsumerAdmin`. This inheritance enables clients to use the `ConsumerAdmin` interface to create pure OMG event service style proxy supplier objects. Proxy supplier objects created in this manner do not support configuration of QoS properties, and do not have associated filters. Proxy suppliers created through the inherited `CosEventChannelAdmin::ConsumerAdmin` interface do not have unique identifiers associated with them, whereas proxy suppliers created by operations supported by the `ConsumerAdmin` interface do have unique identifiers.



---

The `ConsumerAdmin` interface supports a read-only attribute that maintains a reference to the `EventChannel` instance that created it. The `ConsumerAdmin` interface also supports a read-only attribute that contains a unique numeric identifier which is assigned event channel upon creation of a `ConsumerAdmin` instance. This identifier is unique among all `ConsumerAdmin` instances created by a given channel.

As described above, a `ConsumerAdmin` can maintain a list of filters that are applied to all proxy suppliers it creates. Each proxy supplier can also support a list of filters that apply only to the proxy. When combining these two lists during the evaluation of a given event, either `AND` or `OR` semantics may be applied. The choice is determined by an input flag when creating of the `ConsumerAdmin`, and the operator that is used for this purpose by a given `ConsumerAdmin` is maintained in a read-only attribute.

The `ConsumerAdmin` interface also supports attributes that maintain references to priority and lifetime mapping filter objects. These mapping filter objects are applied to all proxy supplier objects created by a given `ConsumerAdmin`.

Each `ConsumerAdmin` assigns a unique numeric identifier to each proxy supplier it maintains. The `ConsumerAdmin` interface supports attributes that maintain the list of these unique identifiers associated with the proxy pull and the proxy push suppliers created by a given `ConsumerAdmin`. The `ConsumerAdmin` interface also supports an operation that, given the unique identifier of a proxy supplier, returns the object reference of that proxy supplier. Finally, the `ConsumerAdmin` interface supports operations that create the various styles of proxy supplier objects supported by the event channel.

## **ConsumerAdmin::MyID**

readonly attribute [AdminID](#) MyID;

Maintains the unique identifier of the target `ConsumerAdmin` instance that is assigned to it upon creation by the event channel.

## **ConsumerAdmin::MyChannel**

readonly attribute [EventChannel](#) MyChannel

---

Maintains the object reference of the event channel that created a given `ConsumerAdmin` instance.

### **ConsumerAdmin::MyOperator**

readonly attribute [InterFilterGroupOperator](#) MyOperator;

Maintains the information regarding whether AND or OR semantics are used during the evaluation of a given event when combining the filter objects associated with the target `ConsumerAdmin` and those defined locally on a given proxy supplier.

### **ConsumerAdmin::priority\_filter**

attribute [CosNotifyFilter::MappingFilter](#) priority\_filter;

Maintains a reference to a mapping filter object that affects how each proxy supplier created by the target `ConsumerAdmin` treats events with respect to priority.

Each proxy supplier also has an associated attribute which maintains a reference to a mapping filter object for the priority property. This local mapping filter object is only used by the proxy supplier in the event that the `priority_filter` attribute of the `ConsumerAdmin` instance that created it is set to `OBJECT_NIL`.

### **ConsumerAdmin::lifetime\_filter**

attribute [CosNotifyFilter::MappingFilter](#) lifetime\_filter;

Maintains a reference to a mapping filter that affects how each proxy supplier created by the target `ConsumerAdmin` treats events with respect to lifetime.

Each proxy supplier object also has an associated attribute that maintains a reference to a mapping filter object for the lifetime property. This local mapping filter object is only used by the proxy supplier in the event that the `lifetime_filter` attribute of the `ConsumerAdmin` instance that created it is set to `OBJECT_NIL`.

---

## ConsumerAdmin::pull\_suppliers

readonly attribute [ProxyIDSeg](#) pull\_suppliers;

Contains the list of unique identifiers that have been assigned by a ConsumerAdmin instance to each pull-style proxy supplier it has created.

## ConsumerAdmin::push\_suppliers

readonly attribute [ProxyIDSeg](#) push\_suppliers;

Contains the list of unique identifiers that have been assigned by a ConsumerAdmin instance to each push-style proxy supplier it has created.

## ConsumerAdmin::get\_proxy\_supplier()

```
ProxySupplier get_proxy_supplier (in ProxyID proxy_id)
  raises ( ProxyNotFound );
```

Returns an object reference to the proxy supplier whose unique id was passed to the method.

### Parameters

`proxy_id` A numeric identifier associated with one of the proxy suppliers that created by the target ConsumerAdmin.

### Exceptions

[ProxyNotFound](#) The input parameter does not correspond to the unique identifier of a proxy supplier object created by the target ConsumerAdmin.

## ConsumerAdmin::obtain\_notification\_pull\_supplier()

```
ProxySupplier obtain_notification_pull_supplier (
  in ClientType ctype,
  out ProxyID proxy_id)
  raises ( AdminLimitExceeded );
```

---

Creates instances of the pull-style proxy suppliers defined in `CosNotifyChannelAdmin` and returns an object reference to the new proxy.

Three varieties of pull-style proxy suppliers are defined in this module:

- The [ProxyPullSupplier](#) interface supports connections to pull consumers that receive events as `Anys`.
- The [StructuredProxyPullSupplier](#) interface supports connections to pull consumers that receive structured events.
- The [SequenceProxyPullSupplier](#) interface support connections to pull consumers that receive sequences of structured events.

The input parameter `flag` indicates which type of pull style proxy instance to create.

The target `ConsumerAdmin` creates the new pull-style proxy supplier and assigns a numeric identifier to it that is unique among all proxy suppliers the `ConsumerAdmin` has created.

## Parameters

<code>ctype</code>	A flag that indicates which style of pull-style proxy supplier to create.
<code>proxy_id</code>	The unique identifier of the new proxy supplier.

## Exceptions

[AdminLimitExceeded](#) The number of consumers currently connected to the channel with which the target `ConsumerAdmin` is associated exceeds the value of the `MaxConsumers` administrative property.

## **ConsumerAdmin::obtain\_notification\_push\_supplier()**

```
ProxySupplier obtain_notification_push_supplier (  
    in ClientType ctype,  
    out ProxyID proxy_id)  
    raises ( AdminLimitExceeded );
```

Creates instances of the push-style proxy supplier objects defined in `CosNotifyChannelAdmin` and returns an object reference to the new proxy.

---

Three varieties of push-style proxy suppliers are defined in this module:

- The [ProxyPushSupplier](#) interface supports connections to push consumers that receive events as `Anys`.
- The [StructuredProxyPushSupplier](#) interface supports connections to push consumers that receive structured events.
- The [SequenceProxyPushSupplier](#) interface supports connections to push consumers that receive sequences of structured events.

The input parameter flag indicates which type of push-style proxy to create.

The target `ConsumerAdmin` creates the new push-style proxy supplier and assigns a numeric identifier to it that is unique among all proxy suppliers the `ConsumerAdmin` has created.

## Parameters

<code>ctype</code>	A flag indicating which style of push-style proxy supplier to create.
<code>proxy_id</code>	The unique identifier of the new proxy supplier.

## Exceptions

[AdminLimitExceeded](#) The number of consumers currently connected to the channel with which the target `ConsumerAdmin` is associated exceeds the value of the `MaxConsumers` administrative property.

## **ConsumerAdmin::destroy()**

```
void destroy();
```

Destroys all proxies under the administration of the target object, and then destroys the target object itself. When destroying each object, it frees any storage associated with the object in question, and then invalidates the object's IOR.



# CosNotifyChannelAdmin:: EventChannel Interface

```
//IDL
interface EventChannel :
    CosNotification::QoSAdmin,
    CosNotification::AdminPropertiesAdmin,
    CosEventChannelAdmin::EventChannel
{
    readonly attribute EventChannelFactory MyFactory;
    readonly attribute ConsumerAdmin default_consumer_admin;
    readonly attribute SupplierAdmin default_supplier_admin;
    readonly attribute CosNotifyFilter::FilterFactory
        default_filter_factory;

    ConsumerAdmin new_for_consumers(
        in InterFilterGroupOperator op,
        out AdminID id );

    SupplierAdmin new_for_suppliers(
        in InterFilterGroupOperator op,
        out AdminID id );

    ConsumerAdmin get_consumeradmin ( in AdminID id )
        raises (AdminNotFound);

    SupplierAdmin get_supplieradmin ( in AdminID id )
        raises (AdminNotFound);

    AdminIDSeq get_all_consumeradmins();
    AdminIDSeq get_all_supplieradmins();
};
```

The EventChannel interface defines the behavior of an event channel. This interface inherits from `CosEventChannelAdmin::EventChannel`; this makes an instance of the notification service EventChannel interface fully compatible with an OMG event service style untyped event channel.

---

Inheritance of `CosEventChannelAdmin::EventChannel` enables an instance of the `EventChannel` interface to create event service style `ConsumerAdmin` and `SupplierAdmin` instances. These instances can subsequently be used to create pure event service style proxies, which support connections to pure event service style suppliers and consumers.

While notification service style proxies and admin objects have unique identifiers associated with them, enabling their references to be obtained by invoking operations on the notification service style admin and event channel interfaces, event service style proxies and admin objects do not have associated unique identifiers, and cannot be returned by invoking an operation on the notification service style admin or event channel interfaces.

The `EventChannel` interface also inherits from the [QoSAdmin](#) and the [AdminPropertiesAdmin](#) interfaces defined in [CosNotification](#). Inheritance of these interfaces enables a notification service style event channel to manage lists of QoS and administrative properties.

The `EventChannel` interface supports a read-only attribute that maintains a reference to the `EventChannelFactory` that created it. Each instance of the `EventChannel` interface has an associated default [ConsumerAdmin](#) and an associated default [SupplierAdmin](#), both of which exist upon creation of the channel and that have the unique identifier of zero. Admin object identifiers must only be unique among a given type of admin, which means that the identifiers assigned to [ConsumerAdmin](#) objects can overlap those assigned to [SupplierAdmin](#) objects. The `EventChannel` interface supports read-only attributes that maintain references to these default admin objects.

The `EventChannel` interface supports operations that create new [ConsumerAdmin](#) and [SupplierAdmin](#) instances. The `EventChannel` interface also supports operations that, when provided with the unique identifier of an admin object, can return references to the [ConsumerAdmin](#) and [SupplierAdmin](#) instances associated with a given `EventChannel`. Finally, the `EventChannel` interface supports operations that return the sequence of unique identifiers of all [ConsumerAdmin](#) and [SupplierAdmin](#) instances associated with a given `EventChannel`.

## EventChannel::MyFactory

```
readonly attribute EventChannelFactory MyFactory;
```



---

Maintains the object reference of the event channel factory that created a given `EventChannel`.

### **EventChannel::default\_consumer\_admin**

readonly attribute [ConsumerAdmin](#) default\_consumer\_admin;

Maintains a reference to the default [ConsumerAdmin](#) associated with the target `EventChannel`. Each `EventChannel` instance has an associated default [ConsumerAdmin](#), that exists upon creation of the channel and is assigned the unique identifier of zero. Clients can create additional event service style [ConsumerAdmin](#) by invoking the inherited `for_consumers` operation, and additional notification service style [ConsumerAdmin](#) by invoking the `new_for_consumers` operation defined by the `EventChannel` interface.

### **EventChannel::default\_supplier\_admin**

readonly attribute [SupplierAdmin](#) default\_supplier\_admin;

Maintains a reference to the default [SupplierAdmin](#) associated with the target `EventChannel`. Each `EventChannel` has an associated default [SupplierAdmin](#), that exists upon creation of the channel and is assigned the unique identifier of zero. Clients can create additional event service style [SupplierAdmin](#) by invoking the inherited `for_suppliers` operation, and additional notification service style [SupplierAdmin](#) by invoking the `new_for_suppliers` operation defined by the `EventChannel` interface.

### **EventChannel::default\_filter\_factory**

readonly attribute [CosNotifyFilter::FilterFactory](#)  
default\_filter\_factory;

Maintains an object reference to the default factory to be used by its associated `EventChannel` for creating filters. If the target channel does not support a default filter factory, the attribute maintains the value of `OBJECT_NIL`.

---

## EventChannel::new\_for\_consumers()

```
ConsumerAdmin new_for_consumers(  
    in InterFilterGroupOperator op,  
    out AdminID id );
```

Creates a notification service style [ConsumerAdmin](#). The new instance is assigned a unique identifier by the target `EventChannel` that is unique among all [ConsumerAdmin](#)s currently associated with the channel. Upon completion, the operation returns the reference to the new [ConsumerAdmin](#), and the unique identifier assigned to the new [ConsumerAdmin](#) as the output parameter.

### Parameters

- |    |   |
|----|---|
| op | A boolean flag indicating whether to use AND or OR semantics when the <a href="#">ConsumerAdmin's</a> filters are combined with the filters associated with any supplier proxies the <a href="#">ConsumerAdmin</a> creates. |
| id | The unique identifier assigned to the new <a href="#">ConsumerAdmin</a> .   |

## EventChannel::new\_for\_suppliers()

```
SupplierAdmin new_for_suppliers(  
    in InterFilterGroupOperator op,  
    out AdminID id );
```

Creates a notification service style [SupplierAdmin](#). The new [SupplierAdmin](#) is assigned an identifier by the target `EventChannel` that is unique among all [SupplierAdmin](#)s currently associated with the channel. Upon completion, the operation returns the reference to the new [SupplierAdmin](#), and the unique identifier assigned to the new [SupplierAdmin](#) as the output parameter.

### Parameters

- |    |   |
|----|---|
| op | A boolean flag indicating whether to use AND or OR semantics when the <a href="#">SupplierAdmin's</a> filters are combined with the filters associated with any supplier proxies the <a href="#">SupplierAdmin</a> creates. |
| id | The unique identifier assigned to the new <a href="#">SupplierAdmin</a> .   |

---

## EventChannel::get\_consumeradmin()

```
ConsumerAdmin get_consumeradmin ( in AdminID id )  
    raises (AdminNotFound);
```

Returns a reference to one of the [ConsumerAdmins](#) associated with the target EventChannel.

---

**Note:** While a notification service event channel can support both event service and notification service style [ConsumerAdmins](#), only notification service style [ConsumerAdmins](#) have unique identifiers.

---

### Parameters

id                      A numeric value that is the unique identifier of one of the [ConsumerAdmins](#) associated with the target EventChannel.

### Exceptions

[AdminNotFound](#)      The id is not the identifier of one of the [ConsumerAdmins](#) associated with the target EventChannel.

## EventChannel::get\_supplieradmin()

```
SupplierAdmin get_supplieradmin ( in AdminID id )  
    raises (AdminNotFound);
```

Returns a reference to one of the [SupplierAdmins](#) associated with the target EventChannel.

---

**Note:** While a notification service style event channel can support both Event service and notification service style [SupplierAdmins](#), only notification service style [SupplierAdmins](#) have unique identifiers.

---

### Parameters

id                      A numeric value that is the unique identifier of one of the [SupplierAdmins](#) associated with the target EventChannel.

---

## Exceptions

[AdminNotFound](#) The `id` is not the unique identifier of one of the `SupplierAdmins` associated with the target `EventChannel`.

### **EventChannel::get\_all\_consumeradmins()**

```
AdminIDSeq get_all_consumeradmins();
```

Returns a sequence of unique identifiers assigned to all notification service style `ConsumerAdmins` created by the target `EventChannel`.

### **EventChannel::get\_all\_supplieradmins()**

```
AdminIDSeq get_all_supplieradmins();
```

Returns a sequence of unique identifiers assigned to all notification service style `SupplierAdmins` created by the target `EventChannel`.

# CosNotifyChannelAdmin:: EventChannelFactory Interface

```
//IDL
interface EventChannelFactory
{
    EventChannel create\_channel (
        in CosNotification::QoSProperties initial_qos,
        in CosNotification::AdminProperties initial_admin,
        out ChannelID id)
        raises(CosNotification::UnsupportedQoS,
            CosNotification::UnsupportedAdmin );

    ChannelIDSeq get\_all\_channels();

    EventChannel get\_event\_channel ( in ChannelID id )
        raises (ChannelNotFound);
};
```

The EventChannelFactory interface defines operations for creating and managing event channels. It supports a routine that creates new instances of event channels and assigns unique numeric identifiers to them.

The EventChannelFactory interface supports a routine that returns the unique identifiers assigned to all event channels created by a given EventChannelFactory, and another routine that, given the unique identifier of an event channel, returns the object reference of that event channel.

## EventChannelFactory::create\_channel()

```
EventChannel create\_channel (
    in CosNotification::QoSProperties initial_qos,
    in CosNotification::AdminProperties initial_admin,
    out ChannelID id)
    raises(CosNotification::UnsupportedQoS,
        CosNotification::UnsupportedAdmin );
```

---

Creates an instance of an event channel and returns an object reference to the new channel.

### Parameters

<code>initial_qos</code>	A list of name-value pairs specifying the initial QoS property settings for the new channel.
<code>initial_admin</code>	A list of name-value pairs specifying the initial administrative property settings for the new channel.
<code>id</code>	A numeric identifier that is assigned to the new event channel and which is unique among all event channels created by the target object.

### Exceptions

[UnsupportedQoS](#) Raised if no implementation of the [EventChannel](#) interface exists that can support all of the requested QoS property settings. This exception contains a sequence of data structures which identifies the name of a QoS property in the input list whose requested setting could not be satisfied, along with an error code and a range of settings for the property that could be satisfied.

[UnsupportedAdmin](#) Raised if no implementation of the [EventChannel](#) interface exists that can support all of the requested administrative property settings. This exception contains a sequence of data structures that identifies the name of an administrative property in the input list whose requested setting could not be satisfied, along with an error code and a range of settings for the property that could be satisfied.

### **EventChannelFactory::get\_all\_channels()**

[ChannelIDSeq](#) `get_all_channels();`

Returns a sequence containing all of the unique numeric identifiers for the event channels which have been created by the target object.

---

## EventChannelFactory::get\_event\_channel()

```
EventChannel get_event_channel ( in ChannelID id )  
    raises ( ChannelNotFound );
```

Returns the object reference of the event channel corresponding to the input identifier.

### Parameters

`id` A numeric value that is the unique identifier of an event channel that has been created by the target object.

### Exceptions

[ChannelNotFound](#) The `id` does not correspond to the unique identifier of an event channel that has been created by the target object.





# CosNotifyChannelAdmin:: ProxyConsumer Interface

```
//IDL in CosNotifyChannelAdmin
interface ProxyConsumer:
    CosNotification::QoSAdmin,
    CosNotifyFilter::FilterAdmin
{
    readonly attribute ProxyType MyType;
    readonly attribute SupplierAdmin MyAdmin;

    CosNotification::EventTypeSeq obtain\_subscription\_types(
        in ObtainInfoMode mode );

    void validate\_event\_qos (
        in CosNotification::QoSProperties required_qos,
        out CosNotification::NamedPropertyRangeSeq available_qos)
    raises(CosNotification::UnsupportedQoS);
};
```

The `ProxyConsumer` interface is an abstract interface that is inherited by the different proxy consumers that can be instantiated within an event channel. It encapsulates the behaviors common to all notification service proxy consumers. In particular, the `ProxyConsumer` interface inherits the [QoSAdmin](#) interface defined within the [CosNotification](#) module, and the [FilterAdmin](#) interface defined within the [CosNotifyFilter](#) module. The former inheritance enables proxy consumers to administer a list of associated QoS properties. The latter inheritance enables proxy consumers to administer a list of associated filter objects. Locally, the `ProxyConsumer` interface defines a read-only attribute that contains a reference to the [SupplierAdmin](#) object that created it. The `ProxyConsumer` interface also defines an operation to return the list of event types a given proxy consumer instance can forward, and an operation to determine which QoS properties can be set on a per-event basis.

---

## ProxyConsumer::obtain\_subscription\_types()

```
CosNotification::EventTypeSeq obtain_subscription_types(  
    in ObtainInfoMode mode);
```

Returns a list of event type names that consumers connected to the channel are interested in receiving.

### Parameters

`mode` Specifies whether to automatically notify the supplier of changes to the subscription list.

## ProxyConsumer::validate\_event\_qos()

```
void validate_event_qos (  
    in CosNotification::QoSProperties required_qos,  
    out CosNotification::NamedPropertyRangeSeq available_qos)  
    raises (CosNotification::UnsupportedQoS);
```

Checks whether the target proxy object will honor the setting of the specified QoS properties on a per-event basis. If all requested QoS property value settings can be satisfied by the target object, the operation returns successfully with an output parameter that contains a sequence of [NamedPropertyRange](#) data structures.

### Parameters

`required_qos` A sequence of QoS property name-value pairs that specify a set of QoS settings that a client is interested in setting on an event.

---

**Note:** The QoS property settings contained in the optional header fields of a structured event may differ from those that are configured on a given proxy object.

---

`available_qos` A sequence of [NamedPropertyRange](#). Each element includes the name of an additional QoS property whose setting is supported by the target object on a per-event basis. Each element also includes the range of values that are acceptable for each property.

---

## Exceptions

[UnsupportedQoS](#) Raised if any of the requested settings cannot be honored by the target object. This exception contains as data a sequence of data structures identifying the name of a QoS property in the input list whose requested setting could not be satisfied, along with an error code and a range of valid settings for the property.

## Exceptions



# CosNotifyChannelAdmin:: ProxyPullConsumer Interface

```
//IDL
interface ProxyPullConsumer :
    ProxyConsumer,
    CosEventComm::PullConsumer
{
    void connect_any_pull_supplier (
        in CosEventComm::PullSupplier pull_supplier)
    raises(CosEventChannelAdmin::AlreadyConnected,
        CosEventChannelAdmin::TypeError);
};
```

The `ProxyPullConsumer` interface supports connections to the channel by suppliers who make events, packaged as `Anys`, available to the channel using the pull model.

The `ProxyPullConsumer` interface extends the OMG event service pull-style suppliers of untyped events by supporting event filtering and the configuration of QoS properties. This interface enables OMG event service style untyped event suppliers to take advantage of the features offered by the notification service.

Through inheritance of the [ProxyConsumer](#) interface, the `ProxyPullConsumer` interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the [SupplierAdmin](#) object that created it. In addition, this inheritance implies that a `ProxyPullConsumer` instance supports an operation that returns the list of event types that consumers connected to the same channel are interested in receiving, and an operation that returns information about the instance's ability to accept a QoS request.

The `ProxyPullConsumer` interface also inherits from the `PullConsumer` interface defined within `CosEventComm`. This interface supports the operation to disconnect the `ProxyPullConsumer` from its associated supplier. Finally, the `ProxyPullConsumer` interface defines the operation to establish the connection over which the pull supplier can send events to the channel.

---

## ProxyPullConsumer::connect\_any\_pull\_supplier()

```
void connect_any_pull_supplier (  
    in CosEventComm::PullSupplier pull_supplier)  
    raises(CosEventChannelAdmin::AlreadyConnected,  
          CosEventChannelAdmin::TypeError);
```

Establishes a connection between a pull-style supplier of events in the form of `Anys`, and the event channel. Once the connection is established, the proxy can proceed to receive events from the supplier by invoking `pull` or `try_pull` on the supplier (whether the proxy invokes `pull` or `try_pull`, and the frequency with which it performs such invocations, is a detail that is specific to the implementation of the channel).

### Parameters

`pull_supplier` A reference to an object supporting the `PullSupplier` interface defined within `CosEventComm`.

### Exceptions

`AlreadyConnected` Raised if the proxy is already connected to a pull supplier.

`TypeError` An implementation of the `ProxyPullConsumer` interface may impose additional requirements on the interface supported by a pull supplier (for example, it may be designed to invoke some operation other than `pull` or `try_pull` in order to receive events). If the pull supplier being connected does not meet those requirements, this operation raises the `TypeError` exception.

# CosNotifyChannelAdmin:: ProxyPullSupplier Interface

```
//IDL
interface ProxyPullSupplier :
    ProxySupplier,
    CosEventComm::PullSupplier
{
    void connect_any_pull_consumer (
        in CosEventComm::PullConsumer pull_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected);
};
```

The `ProxyPullSupplier` interface supports connections to the channel by consumers that pull events from the channel as `Any`s.

The `ProxyPullSupplier` interface extends the OMG event service pull-style consumers of untyped events by supporting event filtering and the configuration of QoS properties. This interface enables OMG event service style untyped event consumers to take advantage of the features offered by the notification service.

Through inheritance of the [ProxySupplier](#) interface, the `ProxyPullSupplier` interface supports administration of QoS properties, administration of a list of associated filter objects, mapping filters for event priority and lifetime, and a read-only attribute containing a reference to the [ConsumerAdmin](#) object that created it. This inheritance also means that a `ProxyPullSupplier` instance supports an operation that returns the list of event types that the proxy supplier will potentially supply, and an operation that returns information about the instance's ability to accept a QoS request.

The `ProxyPullSupplier` interface also inherits from the `PullSupplier` interface defined within the `CosEventComm` module of the OMG event service. This interface supports the `pull` and `try_pull` operations that the consumer connected to a `ProxyPullSupplier` instance invokes to receive an event from the channel in the form of an `Any`, and the operation to disconnect the `ProxyPullSupplier` from its associated consumer.

---

Finally, the `ProxyPullSupplier` interface defines the operation to establish a connection over which the pull consumer receives events from the channel.

### **ProxyPullSupplier::connect\_any\_pull\_consumer()**

```
void connect_any_pull_consumer (  
    in CosEventComm::PullConsumer pull_consumer)  
    raises(CosEventChannelAdmin::AlreadyConnected);
```

Establishes a connection between a pull consumer of events in the form of `Anys` and an event channel. Once established, the consumer can receive events from the channel by invoking `pull` or `try_pull` on its associated `ProxyPullSupplier`.

#### **Parameters**

`pull_consumer` A reference to an object supporting the `PullConsumer` interface defined within the `CosEventComm` module of the OMG event service.

#### **Exceptions**

`AlreadyConnected` The target object of this operation is already connected to a pull consumer object.



# CosNotifyChannelAdmin:: ProxyPushConsumer Interface

```
//IDL
interface ProxyPushConsumer :
    ProxyConsumer,
    CosEventComm::PushConsumer
{
    void connect\_any\_push\_supplier (
        in CosEventComm::PushSupplier push_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected);
};
```

The `ProxyPushConsumer` interface supports connections to the channel by suppliers that push events to the channel as `Any`s.

The `ProxyPushConsumer` extends the OMG event service push consumer interface by supporting event filtering and the configuration of various QoS properties. This interface enables OMG event service style untyped event suppliers to take advantage of these new features offered by the notification service.

Through inheritance of the [ProxyConsumer](#) interface, the `ProxyPushConsumer` interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the `SupplierAdmin` object that created it. In addition, this inheritance means that a `ProxyPushConsumer` instance supports an operation that returns the list of event types that consumers connected to the same channel are interested in receiving, and an operation that returns information about the instance's ability to accept a QoS request.

The `ProxyPushConsumer` interface also inherits from the `PushConsumer` interface defined within the `CosEventComm` module of the OMG event service. This interface supports the `push` operation which the supplier connected to a `ProxyPushConsumer` instance invokes to send an event to the channel in the form of an `Any`, and the operation to disconnect the `ProxyPushConsumer` from its associated supplier.

---

Finally, the `ProxyPushConsumer` interface defines the operation to establish the connection over which the push supplier sends events to the channel.

### **ProxyPushConsumer::connect\_any\_push\_supplier()**

```
void connect_any_push_supplier (  
    in CosEventComm::PushSupplier push_supplier)  
    raises(CosEventChannelAdmin::AlreadyConnected);
```

Establishes a connection between a push-style supplier of events in the form of an `any` and an event channel. Once established, the supplier can send events to the channel by invoking the `push` operation supported by the target `ProxyPushConsumer` instance.

#### **Parameters**

`push_supplier` The reference to an object supporting the `PushSupplier` interface defined within the `CosEventComm` module.

#### **Exceptions**

`AlreadyConnected` The target object of this operation is already connected to a push supplier object.

#### **Exceptions**

# CosNotifyChannelAdmin:: ProxyPushSupplier Interface

```
//IDL
interface ProxyPushSupplier :
    ProxySupplier,
    CosEventComm::PushSupplier
{
    void connect_any_push_consumer (
        in CosEventComm::PushConsumer push_consumer)
    raises(CosEventChannelAdmin::AlreadyConnected,
        CosEventChannelAdmin::TypeError );

    void suspend_connection()
    raises(CosEventChannel::ConnectionAlreadyInactive);

    void resume_connection()
    raises(CosEventChannelAdmin::ConnectionAlreadyActive);
};
```

The `ProxyPushSupplier` interface supports connections to the channel by consumers that receive events from the channel as untyped `Anys`.

The `ProxyPushSupplier` interface extends the OMG event service push-style consumers of untyped events by supporting event filtering and the configuration of QoS properties. Thus, this interface enables OMG event service push-style untyped event consumers to take advantage of the features offered by the notification service.

Through inheritance of [ProxySupplier](#), the `ProxyPushSupplier` interface supports administration of QoS properties, administration of a list of associated filter objects, mapping filters for event priority and lifetime, and a read-only attribute containing a reference to the [ConsumerAdmin](#) that created it. This inheritance also implies that a `ProxyPushSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the instance's ability to accept a QoS request.

---

The `ProxyPushSupplier` interface also inherits from the `PushSupplier` interface defined within `CosEventComm`. This interface supports the operation to disconnect a `ProxyPushSupplier` from its associated consumer.

The `ProxyPushSupplier` interface defines the operation to establish the connection over which the push consumer can receive events from the channel. The `ProxyPushSupplier` interface also defines a pair of operations that can suspend and resume the connection between a `ProxyPushSupplier` and its associated `PushConsumer`. During the time a connection is suspended, the `ProxyPushSupplier` accumulates events destined for the consumer but does not transmit them until the connection is resumed.

### **ProxyPushSupplier::connect\_any\_push\_consumer()**

```
void connect_any_push_consumer (  
    in CosEventComm::PushConsumer push_consumer)  
raises(CosEventChannelAdmin::AlreadyConnected,  
    CosEventChannelAdmin::TypeError );
```

Establishes a connection between a push-style consumer of events in the form of `Anys`, and the event channel. Once the connection is established, the `ProxyPushSupplier` sends events to its associated consumer by invoking `push` on the consumer.

#### **Parameters**

`push_consumer` A reference to an object supporting the `PushConsumer` interface defined within `CosEventComm`

#### **Exceptions**

`AlreadyConnected` Raised if the proxy is already connected to a push consumer.

`TypeError` An implementation of the `ProxyPushSupplier` interface may impose additional requirements on the interface supported by a push consumer (for example, it may be designed to invoke some operation other than `push` in order to transmit events). If the push consumer being connected does not meet those requirements, this operation raises the `TypeError` exception.

---

## ProxyPushSupplier::suspend\_connection()

```
void suspend_connection()  
    raises(ConnectionAlreadyInactive);
```

Causes the `ProxyPushSupplier` to stop sending events to the `PushConsumer` instance connected to it. The `ProxyPushSupplier` does not forward events to its associated `PushConsumer` until `resume_connection()` is invoked. During this time, the `ProxyPushSupplier` continues to queue events destined for the `PushConsumer`; however, events that time out prior to resumption of the connection are discarded. Upon resumption of the connection, all queued events are forwarded to the `PushConsumer`.

**Exceptions** The [ConnectionAlreadyInactive](#) exception is raised if the connection is currently in a suspended state.

## ProxyPushSupplier::resume\_connection()

```
void resume_connection()  
    raises(ConnectionAlreadyActive);
```

Causes the `ProxyPushSupplier` interface to resume sending events to the `PushConsumer` instance connected to it, including those events that have been queued while the connection was suspended and have not yet timed out.

**Exceptions** [ConnectionAlreadyActive](#) The connection is not in a suspended state.



# CosNotifyChannelAdmin:: ProxySupplier Interface

```
//IDL
interface ProxySupplier :
    CosNotification::QoSAdmin,
    CosNotifyFilter::FilterAdmin
{
    readonly attribute ConsumerAdmin MyAdmin;
    readonly attribute ProxyType MyType;
    attribute CosNotifyFilter::MappingFilter priority_filter;
    attribute CosNotifyFilter::MappingFilter lifetime_filter;

    CosNotification::EventTypeSeq obtain_offered_types(
        in ObtainInfoMode mode );

    void validate\_event\_qos (
        in CosNotification::QoSProperties required_qos,
        out CosNotification::NamedPropertyRangeSeq available_qos)
        raises (CosNotification::UnsupportedQoS);
};
```

The `ProxySupplier` interface is an abstract interface that is inherited by the different proxy suppliers that can be instantiated within an event channel. It encapsulates the behaviors common to all notification service proxy suppliers. In particular, the `ProxySupplier` interface inherits the [QoSAdmin](#) interface defined within the [CosNotification](#) module, and the [FilterAdmin](#) interface defined within the [CosNotifyFilter](#) module. The former inheritance enables proxy suppliers to administer a list of associated QoS properties. The latter inheritance enables proxy suppliers to administer a list of associated filter objects.

Locally, the `ProxySupplier` interface defines a read-only attribute that contains a reference to the [ConsumerAdmin](#) object that created it. In addition, the `ProxySupplier` interface defines attributes that associate two mapping

---

filter objects with each proxy supplier, one for priority and one for lifetime. For more information on mapping filters refer to the *CORBA Notification Service Guide*.

Lastly, the `ProxySupplier` interface defines an operation to return the list of event types that a given proxy supplier can forward to its associated consumer, and an operation to determine which QoS properties can be set on a per-event basis.

### **ProxySupplier::priority\_filter**

attribute [CosNotifyFilter::MappingFilter](#) `priority_filter`;

Contains a reference to an object supporting the [MappingFilter](#) interface defined in the [CosNotifyFilter](#) module. Such an object encapsulates a list of constraint-value pairs, where each constraint is a boolean expression based on the type and contents of an event, and the value is a possible priority setting for the event.

Upon receipt of an event by a proxy supplier object whose `priority_filter` attribute contains a non-zero reference, the proxy supplier invokes the `match` operation supported by the mapping filter object. The mapping filter object then applies its encapsulated constraints to the event.

If the `match` operation returns `TRUE`, the proxy supplier changes the events priority to the value specified in the constraint-value pair that matched the event.

If the `match` operation returns `FALSE`, the proxy supplier checks if the events priority property is already set. If so, the filter does nothing. If the priority property is not set, the filter sets the priority property to its default value.

### **ProxySupplier::lifetime\_filter**

attribute [CosNotifyFilter::MappingFilter](#) `lifetime_filter`;

Contains a reference to an object supporting the [MappingFilter](#) interface defined in the [CosNotifyFilter](#) module. Such an object encapsulates a list of constraint-value pairs, where each constraint is a boolean expression based on the type and contents of an event, and the value is a possible lifetime setting for the event.



---

Upon receipt of each event by a proxy supplier object whose `lifetime_filter` attribute contains a non-zero reference, the proxy supplier invokes the `match` operation supported by the mapping filter object. The mapping filter object then proceeds to apply its encapsulated constraints to the event.

If the `match` operation returns `TRUE`, the proxy supplier changes the events lifetime to the value specified in the constraint-value pair that matched the event.

If the `match` operation returns `FALSE`, the proxy supplier checks if the events lifetime property is already set. If so, the filter does nothing. If the lifetime property is not set, the filter sets the lifetime property to its default value.

## ProxySupplier::obtain\_offered\_types()

```
CosNotification::EventTypeSeq obtain_offered_types(  
    in ObtainInfoMode mode );
```

Returns a list names of event types that the target proxy supplier can forward to its associated consumer.

This mechanism relies on event suppliers keeping the channel informed of the types of events they plan to supply by invoking the `offer_change` operation on their associated proxy consumer objects. The proxy consumers automatically share the information about supplied event types with the proxy suppliers associated with the channel. This enables consumers to discover the types of events that can be supplied to them by the channel by invoking the `obtain_offered_types` operation on their associated proxy supplier.

### Parameters

<code>mode</code>	Specifies how to notify consumers of changes to the publication list.
-------------------	---

## ProxySupplier::validate\_event\_qos()

```
void validate_event_qos (  
    in CosNotification::QoSProperties required_qos,
```

---

```
    out CosNotification::NamedPropertyRangeSeq available_qos)
    raises (CosNotification::UnsupportedQoS);
```

Checks whether the target proxy object will honor the setting of the specified QoS properties on a per-event basis. If all requested QoS property value settings can be satisfied by the target object, the operation returns successfully with an output parameter that contains a sequence of [NamedPropertyRange](#) data structures.

## Parameters

`required_qos` A sequence of QoS property name-value pairs that specify a set of QoS settings that a client is interested in setting on an event

---

**Note:** The QoS property settings contained in the optional header fields of a structured event may differ from those that are configured on a given proxy object.

---

`available_qos` A sequence of [NamedPropertyRange](#). Each element includes the name of a an additional QoS property whose setting is supported by the target object on a per-event basis. Each element also includes the range of values that are acceptable for each such property.

## Exceptions

[UnsupportedQoS](#) Raised if any of the requested settings cannot be honored by the target object. This exception contains as data a sequence of data structures, each of which identifies the name of a QoS property in the input list whose requested setting could not be satisfied, along with an error code and a range of settings for the property that could be satisfied.

# CosNotifyChannelAdmin:: SequenceProxyPullConsumer Interface

```
//IDL
interface SequenceProxyPullConsumer :
    ProxyConsumer,
    CosNotifyComm::SequencePullConsumer
{
    void connect_sequence_pull_supplier (
        in CosNotifyComm::SequencePullSupplier pull_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected,
            CosEventChannelAdmin::TypeError );
};
```

The `SequenceProxyPullConsumer` interface supports connections to the channel by suppliers who make sequences of structured events available to the channel using the pull model.

Through inheritance of [ProxyConsumer](#), the `SequenceProxyPullConsumer` interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the [SupplierAdmin](#) that created it. This inheritance also implies that a `SequenceProxyPullConsumer` supports an operation that returns the list of event types that consumers connected to the same channel are interested in receiving, and an operation that returns information about the instance's ability to accept a QoS request.

The `SequenceProxyPullConsumer` interface also inherits from the [SequencePullConsumer](#) interface defined in the [CosNotifyComm](#) module. This interface supports the operation to close the connection from the supplier to the `SequenceProxyPullConsumer`. Since the [SequencePullConsumer](#) interface inherits from `NotifyPublish`, a supplier can inform its associated `SequenceProxyPullConsumer` whenever the list of event types it plans to supply to the channel changes.

---

The `SequenceProxyPullConsumer` interface also defines a method to establish a connection between the supplier and an event channel.

### **SequenceProxyPullConsumer:: connect\_sequence\_pull\_supplier()**

```
void connect_sequence_pull_supplier (  
    in CosNotifyComm::SequencePullSupplier pull_supplier)  
    raises(CosEventChannelAdmin::AlreadyConnected,  
          CosEventChannelAdmin::TypeError );
```

Establishes a connection between a pull-style supplier of sequences of structured events and the event channel. Once the connection is established, the proxy can receive events from the supplier by invoking `pull_structured_events` or `try_pull_structured_events` on the supplier (whether the proxy invokes `pull_structured_events` or `try_pull_structured_events`, and the frequency with which it performs such invocations, is a detail specific to the implementation of the channel).

#### **Parameters**

`pull_supplier`    A reference to an object supporting the [SequencePullSupplier](#) interface defined within [CosNotifyComm](#).

#### **Exceptions**

`AlreadyConnected` Raised if the proxy is already connected to a pull supplier.

`TypeError`        An implementation of the `SequenceProxyPullConsumer` interface may impose additional requirements on the interface supported by a pull supplier (for example, it may be designed to invoke some operation other than `pull_structured_events` or `try_pull_structured_events` in order to receive events). If the pull supplier being connected does not meet those requirements, this operation raises the `TypeError` exception.

# CosNotifyChannelAdmin:: SequenceProxyPushConsumer Interface

```
//IDL
interface SequenceProxyPushConsumer :
    ProxyConsumer,
    CosNotifyComm::SequencePushConsumer
{
    void connect_sequence_push_supplier (
        in CosNotifyComm::SequencePushSupplier push_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected);
};
```

The `SequenceProxyPushConsumer` interface supports connections to the channel by suppliers that push events to the channel as sequences of structured events.

Through inheritance of the [ProxyConsumer](#) interface, the interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the [SupplierAdmin](#) object that created it. In addition, this inheritance means that a `SequenceProxyPushConsumer` instance supports an operation that returns the list of event types that consumers connected to the same channel are interested in receiving, and an operation that returns information about the instance's ability to accept a QoS request.

The `SequenceProxyPushConsumer` interface also inherits from the [SequencePushConsumer](#) interface defined in the [CosNotifyComm](#) module. This interface supports the operation that enables a supplier of sequences of structured events to push them to a `SequenceProxyPushConsumer`, and also the operation to close down the connection from the supplier to the `SequenceProxyPushConsumer`. Since the [SequencePushConsumer](#) interface inherits from the [NotifyPublish](#) interface, a supplier can inform its associated `SequenceProxyPushConsumer` when the list of event types it supplies to the channel changes.

---

Lastly, the `SequenceProxyPushConsumer` interface defines a method to establish a connection between a supplier and an event channel.

### **SequenceProxyPushConsumer:: connect\_sequence\_push\_supplier()**

```
void connect_sequence_push_supplier (  
    in CosNotifyComm::SequencePushSupplier push_supplier)  
    raises(CosEventChannelAdmin::AlreadyConnected);
```

Establishes a connection between a push-style supplier of sequences of structured events and an event channel. Once the connection is established, the supplier can send events to the channel by invoking `push_structured_events` on its associated `SequenceProxyPushConsumer`.

#### **Parameters**

`push_supplier` A reference to an object supporting the [SequencePushSupplier](#) interface defined within the [CosNotifyComm](#) module.

#### **Exceptions**

`AlreadyConnected`The proxy is already connected to a push supplier object.

# CosNotifyChannelAdmin:: SequenceProxyPullSupplier Interface

```
//IDL
interface SequenceProxyPullSupplier :
    ProxySupplier,
    CosNotifyComm::SequencePullSupplier
{
    void connect_sequence_pull_consumer (
        in CosNotifyComm::SequencePullConsumer pull_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected);
};
```

The `SequenceProxyPullSupplier` interface supports connections to the channel by consumers who pull sequences of structured events from an event channel.

Through inheritance of the [ProxySupplier](#) interface, the `SequenceProxyPullSupplier` interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the [ConsumerAdmin](#) object that created it. In addition, this inheritance implies that a `SequenceProxyPullSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the instance's ability to accept a QoS request.

The `SequenceProxyPullSupplier` interface also inherits from the [SequencePullSupplier](#) interface defined in [CosNotifyComm](#). This interface supports the operations enabling a consumer of sequences of structured events to pull them from the `SequenceProxyPullSupplier`, and also the operation to close the connection from the consumer to its associated `SequenceProxyPullSupplier`. Since the [SequencePullSupplier](#) interface inherits from the [NotifySubscribe](#) interface, a `SequenceProxyPullSupplier` can be notified whenever the list of event types that its associated consumer is interested in receiving changes.

The `SequenceProxyPullSupplier` interface also defines a method to establish a connection between the consumer and an event channel.

---

## SequenceProxyPullSupplier:: connect\_sequence\_pull\_consumer()

```
void connect_sequence_pull_consumer (  
    in CosNotifyComm::SequencePullConsumer pull_consumer)  
    raises(CosEventChannelAdmin::AlreadyConnected);
```

Establishes a connection between a pull-style consumer of sequences of structured events and the event channel. Once the connection is established, the consumer can proceed to receive events from the channel by invoking `pull_structured_events` or `try_pull_structured_events` on its associated `SequenceProxyPullSupplier`.

### Parameters

`pull_consumer`     A reference to an object supporting the [SequencePullConsumer](#) interface defined in [CosNotifyComm](#).

### Exceptions

`AlreadyConnected` The proxy is already connected to a pull consumer.



# CosNotifyChannelAdmin::SequenceProxyPushSupplier Interface

```
//IDL
interface SequenceProxyPushSupplier :
    ProxySupplier,
    CosNotifyComm::SequencePushSupplier
{
    void connect_sequence_push_consumer (
        in CosNotifyComm::SequencePushConsumer push_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected,
            CosEventChannelAdmin::TypeError );

    void suspend_connection()
        raises(ConnectionAlreadyInactive);

    void resume_connection()
        raises(ConnectionAlreadyActive);
};
```

The `SequenceProxyPushSupplier` interface supports connections to the channel by consumers that receive sequences of structured events from the channel.

Through inheritance of [ProxySupplier](#), the `SequenceProxyPushSupplier` interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the [ConsumerAdmin](#) that created it. This inheritance also implies that a `SequenceProxyPushSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the instance's ability to accept a QoS request.

The `SequenceProxyPushSupplier` interface also inherits from the [SequencePushSupplier](#) interface defined in [CosNotifyComm](#). This interface supports the operation to close the connection from the consumer to the `SequenceProxyPushSupplier`. Since the [SequencePushSupplier](#) interface

---

inherits from the [NotifySubscribe](#) interface, a `SequenceProxyPushSupplier` can be notified whenever the list of event types that its associated consumer is interested in receiving changes.

Lastly, the `SequenceProxyPushSupplier` interface defines the operation to establish the connection over which the push consumer receives events from the channel. The `SequenceProxyPushSupplier` interface also defines a pair of operations to suspend and resume the connection between a `SequenceProxyPushSupplier` instance and its associated `SequencePushConsumer`. While a connection is suspended, the `SequenceProxyPushSupplier` accumulates events destined for the consumer but does not transmit them until the connection is resumed.

### **SequenceProxyPushSupplier:: connect\_sequence\_push\_consumer()**

```
void connect_sequence_push_consumer (
    in CosNotifyComm::SequencePushConsumer push_consumer)
    raises(CosEventChannelAdmin::AlreadyConnected,
          CosEventChannelAdmin::TypeError );
```

Establishes a connection between a push-style consumer of sequences of structured events and the event channel. Once the connection is established, the `SequenceProxyPushSupplier` sends events to its associated consumer by invoking `push_structured_events`.

#### **Parameters**

`push_consumer`    A reference to a [SequencePushConsumer](#).

---

## Exceptions

<code>AlreadyConnected</code>	Raised if the proxy is already connected to a push consumer.
<code>TypeError</code>	An implementation of the <code>SequenceProxyPushSupplier</code> interface may impose additional requirements on the interface supported by a push consumer (for example, it may be designed to invoke some operation other than <code>push_structured_events</code> in order to transmit events). If the push consumer being connected does not meet those requirements, this operation raises the <code>TypeError</code> exception.

### `SequenceProxyPushSupplier::suspend_connection()`

```
void suspend_connection()  
    raises(ConnectionAlreadyInactive);
```

Causes the `SequenceProxyPushSupplier` to stop sending events to the `PushConsumer` instance connected to it. The `StructuredProxyPushSupplier` does not forward events to its [SequencePushConsumer](#) until `resume_connection()` is invoked. During this time, the `SequenceProxyPushSupplier` continues to queue events destined for the [SequencePushConsumer](#); however, events that time out prior to resumption of the connection are discarded. Upon resumption of the connection, all queued events are forwarded to the [SequencePushConsumer](#).

## Exceptions

[ConnectionAlreadyInactive](#) The connection is already suspended.

### `SequenceProxyPushSupplier::resume_connection()`

```
void resume_connection()  
    raises(ConnectionAlreadyActive);
```

Causes the `SequenceProxyPushSupplier` to resume sending events to the [SequencePushConsumer](#) instance connected to it, including those that have been queued while the connection was suspended and have not yet timed out.

---

## Exceptions

[ConnectionAlreadyActive](#) The connection is not suspended.

# CosNotifyChannelAdmin:: StructuredProxyPullConsumer Interface

```
//IDL
interface StructuredProxyPullConsumer :
    ProxyConsumer,
    CosNotifyComm::StructuredPullConsumer
{
    void connect_structured_pull_supplier (
        in CosNotifyComm::StructuredPullSupplier pull_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected,
            CosEventChannelAdmin::TypeError );
};
```

The `StructuredProxyPullConsumer` interface supports connections to the channel by suppliers that make structured events available to the channel using the pull model.

Through inheritance of [ProxyConsumer](#), the `StructuredProxyPullConsumer` interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the [SupplierAdmin](#) object that created it. This inheritance also implies that a `StructuredProxyPullConsumer` instance supports an operation that returns the list of event types that consumers connected to the same channel are interested in receiving, and an operation that returns information about the instance's ability to accept a QoS request.

The `StructuredProxyPullConsumer` interface also inherits from the [StructuredPullConsumer](#) interface defined in [CosNotifyComm](#). This interface supports the operation to close the connection from the supplier to the `StructuredProxyPullConsumer`. Since the [StructuredPullConsumer](#) interface inherits from `NotifyPublish`, a supplier can inform the `StructuredProxyPullConsumer` to which it is connected whenever the list of event types it plans to supply to the channel changes.

---

Lastly, the `StructuredProxyPullConsumer` interface defines a method to establish a connection between the supplier and an event channel.

### **StructuredProxyPullConsumer:: connect\_structured\_pull\_supplier()**

```
void connect_structured_pull_supplier (  
    in CosNotifyComm::StructuredPullSupplier pull_supplier)  
    raises(CosEventChannelAdmin::AlreadyConnected,  
          CosEventChannelAdmin::TypeError );
```

Establishes a connection between a pull-style supplier of structured events and the event channel. Once the connection is established, the proxy can receive events from the supplier by invoking `pull_structured_event` or `try_pull_structured_event` on the supplier (whether the proxy invokes `pull_structured_event` or `try_pull_structured_event`, and the frequency with which it performs such invocations, is a detail specific to the implementation of the channel).

#### **Parameters**

`pull_supplier`    A reference to an object supporting the [StructuredPullSupplier](#) interface defined within [CosNotifyComm](#).

#### **Exceptions**

`AlreadyConnected`    Raised if the proxy is already connected to a pull supplier.

`TypeError`            An implementation of the `StructuredProxyPullConsumer` interface may impose additional requirements on the interface supported by a pull supplier (for example, it may be designed to invoke some operation other than `pull_structured_event` or `try_pull_structured_event` in order to receive events). If the pull supplier being connected does not meet those requirements, this operation raises the `TypeError` exception.

# CosNotifyChannelAdmin:: StructuredProxyPullSupplier Interface

```
//IDL
interface StructuredProxyPullSupplier :
    ProxySupplier,
    CosNotifyComm::StructuredPullSupplier
{
    void connect_structured_pull_consumer (
        in CosNotifyComm::StructuredPullConsumer pull_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected);
};
```

The `StructuredProxyPullSupplier` interface supports connections to the channel by consumers that pull structured events from the channel.

Through inheritance of [ProxySupplier](#), the `StructuredProxyPullSupplier` interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the [ConsumerAdmin](#) object that created it. In addition, this inheritance means that a `StructuredProxyPullSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the instance's ability to accept a QoS request.

The `StructuredProxyPullSupplier` interface also inherits from the [StructuredPullSupplier](#) interface defined in [CosNotifyComm](#). This interface supports the operations enabling a consumer of structured events to pull them from a `StructuredProxyPullSupplier`, and the operation to close the connection from the consumer to the `StructuredProxyPullSupplier`. Since the [StructuredPullSupplier](#) interface inherits from [NotifySubscribe](#), a `StructuredProxyPullSupplier` can be notified whenever the list of event types that its associated consumer is interested in receiving changes.

Lastly, the `StructuredProxyPullSupplier` interface defines a method to establish a connection between the consumer and an event channel.

---

## StructuredProxyPullSupplier:: connect\_structured\_pull\_consumer()

```
void connect_structured_pull_consumer (  
    in CosNotifyComm::StructuredPullSupplier pull_consumer)  
    raises(CosEventChannelAdmin::AlreadyConnected);
```

Establishes a connection between a pull consumer of structured events and the event channel. Once established, the consumer can receive events from the channel by invoking `pull_structured_event` or `try_pull_structured_event` on its associated `StructuredProxyPullSupplier`.

### Parameters

`pull_consumer`    A reference to an object supporting the [StructuredPullSupplier](#) interface defined in [CosNotifyComm](#).

### Exceptions

`AlreadyConnected` The proxy is already connected to a pull consumer.



# CosNotifyChannelAdmin:: StructuredProxyPushConsumer Interface

```
//IDL
interface StructuredProxyPushConsumer :
    ProxyConsumer,
    CosNotifyComm::StructuredPushConsumer
{
    void connect_structured_push_supplier (
        in CosNotifyComm::StructuredPushSupplier push_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected);
};
```

The `StructuredProxyPushConsumer` interface supports connections to the channel by suppliers that push events to the channel as structured events.

Through inheritance of the [ProxyConsumer](#) interface, the interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the [SupplierAdmin](#) object that created it. In addition, this inheritance means that a `StructuredProxyPushConsumer` instance supports an operation that returns the list of event types that consumers connected to the same channel are interested in receiving, and an operation that returns information about the instance's ability to accept a QoS request.

The `StructuredProxyPushConsumer` interface also inherits from the [StructuredPushConsumer](#) interface defined in the [CosNotifyComm](#) module. This interface supports the operation that enables a supplier of structured events to push them to the `StructuredProxyPushConsumer`, and also an operation to close down the connection from the supplier to the `StructuredProxyPushConsumer`. Since the [StructuredPushConsumer](#) interface inherits from the [NotifyPublish](#) interface, a supplier can inform the `StructuredProxyPushConsumer` to which it is connected whenever the list of event types it plans to supply to the channel changes.

---

Lastly, the `StructuredProxyPushConsumer` interface defines a method to establish a connection between the supplier and an event channel.

### **StructuredProxyPushConsumer:: connect\_structured\_push\_supplier()**

```
void connect_structured_push_supplier (  
    in CosNotifyComm::StructuredPushSupplier push_supplier)  
    raises(CosEventChannelAdmin::AlreadyConnected);
```

Establishes a connection between a push-style supplier of structured events and the event channel. Once the connection is established, the supplier can send events to the channel by invoking `push_structured_event` on its associated `StructuredProxyPushConsumer` instance.

#### **Parameters**

`push_supplier`A reference to an object supporting the [StructuredPushSupplier](#) interface defined within the [CosNotifyComm](#) module.

#### **Exceptions**

`AlreadyConnected`The proxy object is already connected to a push supplier object.

#### **Exceptions**

# CosNotifyChannelAdmin:: StructuredProxyPushSupplier Interface

```
//IDL
interface StructuredProxyPushSupplier :
    ProxySupplier,
    CosNotifyComm::StructuredPushSupplier
{

    void connect_structured_push_consumer (
        in CosNotifyComm::StructuredPushConsumer push_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected,
            CosEventChannelAdmin::TypeError );

    void suspend_connection()
        raises(ConnectionAlreadyInactive);

    void resume_connection()
        raises(ConnectionAlreadyActive);
};
```

The `StructuredProxyPushSupplier` interface supports connections to the channel by consumers that receive structured events from the channel.

Through inheritance of [ProxySupplier](#), the `StructuredProxyPushSupplier` interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the [ConsumerAdmin](#) that created it. This inheritance also implies that a `StructuredProxyPushSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the instance's ability to accept a QoS request.

The `StructuredProxyPushSupplier` interface also inherits from the [StructuredPushSupplier](#) interface defined in [CosNotifyComm](#). This interface supports the operation that to close the connection from the consumer to the

---

`StructuredProxyPushSupplier`. Since [StructuredPushSupplier](#) inherits from [NotifySubscribe](#), a `StructuredProxyPushSupplier` can be notified whenever the list of event types that its associated consumer is interested in receiving changes.

Lastly, the `StructuredProxyPushSupplier` interface defines the operation to establish the connection over which the push consumer can receive events from the channel. The `StructuredProxyPushSupplier` interface also defines a pair of operations to suspend and resume the connection between a `StructuredProxyPushSupplier` and its associated `StructuredPushConsumer`. During the time such a connection is suspended, the `StructuredProxyPushSupplier` accumulates events destined for the consumer but does not transmit them until the connection is resumed.

### **StructuredProxyPushSupplier:: connect\_structured\_push\_consumer()**

```
void connect_structured_push_consumer (  
    in CosNotifyComm::StructuredPushConsumer push_consumer)  
    raises(CosEventChannelAdmin::AlreadyConnected,  
          CosEventChannelAdmin::TypeError );
```

Establishes a connection between a push-style consumer of structured events and the event channel. Once the connection is established, the `StructuredProxyPushSupplier` sends events to the consumer by invoking `push_structured_event`.

#### **Parameters**

`push_consumer`    A reference to an object supporting the [StructuredPushConsumer](#) interface defined within [CosNotifyComm](#)

#### **Exceptions**

`AlreadyConnectedRaised` if the proxy is already connected to a push consumer.

---

`TypeError`

An implementation of the `StructuredProxyPushSupplier` interface may impose additional requirements on the interface supported by a push consumer (for example, it may be designed to invoke some operation other than `push_structured_event` to transmit events). If the push consumer being connected does not meet those requirements, this operation raises the `TypeError` exception.

### **StructuredProxyPushSupplier::suspend\_connection()**

```
void suspend_connection()  
    raises(ConnectionAlreadyInactive);
```

Causes the `StructuredProxyPushSupplier` to stop sending events to the `PushConsumer` connected to it. The `StructuredProxyPushSupplier` does not forward events to its `StructuredPushConsumer` until `resume_connection()` is invoked. During this time, the `StructuredProxyPushSupplier` queues events destined for the `StructuredPushConsumer`; however, events that time out prior to resumption of the connection are discarded. Upon resumption of the connection, all queued events are forwarded to the `StructuredPushConsumer`.

#### **Exceptions**

[ConnectionAlreadyInactive](#) The connection is already suspended.

### **StructuredProxyPushSupplier::resume\_connection()**

```
void resume_connection()  
    raises(ConnectionAlreadyActive);
```

Causes causes the `StructuredProxyPushSupplier` to resume sending events to the `StructuredPushConsumer` connected to it, including those that have been queued while the connection was suspended and have not yet timed out.

#### **Exceptions**

[ConnectionAlreadyActive](#) The connection is not currently suspended.



# CosNotifyChannelAdmin:: SupplierAdmin Interface

```
//IDL
interface SupplierAdmin :
    CosNotification::QoSAdmin,
    CosNotifyComm::NotifyPublish,
    CosNotifyFilter::FilterAdmin,
    CosEventChannelAdmin::SupplierAdmin
{
    readonly attribute AdminID MyID;
    readonly attribute EventChannel MyChannel;

    readonly attribute InterFilterGroupOperator MyOperator;

    readonly attribute ProxyIDSeq pull_consumers;
    readonly attribute ProxyIDSeq push_consumers;

    ProxyConsumer get\_proxy\_consumer(in ProxyID proxy_id )
        raises ( ProxyNotFound );

    ProxyConsumer obtain\_notification\_pull\_consumer (
        in ClientType ctype,
        out ProxyID proxy_id)
        raises ( AdminLimitExceeded );

    ProxyConsumer obtain\_notification\_push\_consumer (
        in ClientType ctype,
        out ProxyID proxy_id)
        raises ( AdminLimitExceeded );

    ProxyConsumer obtain\_txn\_notification\_push\_consumer (
        in ClientType ctype,
        out ProxyID proxy_id)
        raises ( AdminLimitExceeded );

    void destroy();
```

---

```
};
```

The `SupplierAdmin` interface defines the behavior of objects that create and manage lists of proxy consumers within an event channel. A event channel can have any number of `SupplierAdmin` instances associated with it. Each instance is responsible for creating and managing a list of proxy consumers that share a common set of QoS property settings, and a common set of filters. This feature enables clients to group proxy consumer objects within a channel into groupings that each support a set of suppliers with a common set of QoS requirements, and that make event forwarding decisions using a common set of filters.

The `SupplierAdmin` interface inherits [QoSAdmin](#). This enables each `SupplierAdmin` to manage a set of QoS property settings. These QoS property settings are assigned as the default QoS property settings for any proxy consumer created by a `SupplierAdmin`.

The `SupplierAdmin` interface inherits from the [FilterAdmin](#) interface defined in [CosNotifyFilter](#), enabling each `SupplierAdmin` to maintain a list of filters. These filters encapsulate subscriptions that apply to all proxy consumer objects that have been created by a given `SupplierAdmin` instance.

The `SupplierAdmin` interface also inherits from the [NotifyPublish](#) interface defined in [CosNotifyComm](#). This inheritance enables a `SupplierAdmin` to be the target of an [offer\\_change](#) request made by a supplier, and for the change in event types being offered to be shared by all proxy consumer that were created by the target `SupplierAdmin`. This optimizes the notification of a group of proxy consumers that have been created by the same `SupplierAdmin` of changes to the types of events being offered by suppliers.

The `SupplierAdmin` interface also inherits from `CosEventChannelAdmin::SupplierAdmin`. This inheritance enables clients to use the `SupplierAdmin` interface to create pure OMG event service style proxy consumer objects. Proxy consumer objects created in this manner do not support configuration of QoS properties, and do not have associated filters. Proxy consumer objects created through the inherited `CosEventChannelAdmin::SupplierAdmin` interface do not have unique identifiers associated with them, whereas proxy consumers created by invoking the operations supported by the `SupplierAdmin` interface do.

The `SupplierAdmin` interface supports a read-only attribute that maintains a reference to the `EventChannel` that created a given `SupplierAdmin`. The `SupplierAdmin` interface also supports a read-only attribute that contains a



---

numeric identifier that is assigned to a `SupplierAdmin` the event channel that creates it. This identifier is unique among all `SupplierAdmins` created by a given channel.

A `SupplierAdmin` maintains a list of filters that are applied to all proxy consumers it creates. Each proxy consumer also supports a list of filters that apply only that proxy. When combining these two lists during the evaluation of an event, either `AND` or `OR` semantics can be applied. The choice is determined by an input flag upon creation of the `SupplierAdmin`, and the operator that is used for this purpose by a given `SupplierAdmin` is maintained in a read-only attribute.

Each `SupplierAdmin` assigns a unique numeric identifier to each proxy consumer it maintains. The `SupplierAdmin` interface supports attributes that maintain the list of these unique identifiers associated with the proxy pull and the proxy push consumers created by a given `SupplierAdmin`. The `SupplierAdmin` interface also supports an operation which, when provided with the unique identifier of a proxy consumer, returns the object reference of that proxy consumer object. Finally, the `SupplierAdmin` interface supports operations that can create the various styles of proxy consumers supported by the event channel.

### **SupplierAdmin::MyID**

readonly attribute [AdminID](#) MyID;

Maintains the unique identifier of the target `SupplierAdmin`. This ID is assigned to it upon creation by the event channel.

### **SupplierAdmin::MyChannel**

readonly attribute `EventChannel` MyChannel;

Maintains an object reference to the event channel that created the `SupplierAdmin`.

---

## SupplierAdmin::MyOperator

readonly attribute [InterFilterGroupOperator](#) MyOperator;;

Maintains the information regarding whether AND or OR semantics are used during the evaluation of events when combining the filters associated with the target `SupplierAdmin` and those defined on a given proxy consumer.

## SupplierAdmin::pull\_consumers

readonly attribute [ProxyIDSeq](#) pull\_consumers;

Contains the list of unique identifiers assigned by a `SupplierAdmin` to each pull-style proxy consumer it has created.

## SupplierAdmin::push\_consumers

readonly attribute [ProxyIDSeq](#) push\_consumers;

Contains the list of unique identifiers assigned by a `SupplierAdmin` to each push-style proxy consumer it has created.

## SupplierAdmin::get\_proxy\_consumer()

```
ProxyConsumer get_proxy_consumer ( in ProxyID proxy_id )  
raises ( ProxyNotFound );
```

Returns an object reference to the proxy consumer whose unique identifier was specified.

### Parameters

`proxy_id`            The numeric identifier associated with one of the proxy consumers created by the target `SupplierAdmin`.

### Exceptions

[ProxyNotFound](#)    The input parameter does not correspond to the unique identifier of a proxy consumer created by the target `SupplierAdmin`.

---

## SupplierAdmin::obtain\_notification\_pull\_consumer()

```
ProxyConsumer obtain_notification_pull_consumer (  
    in ClientType ctype,  
    out ProxyID proxy_id)  
    raises ( AdminLimitExceeded );
```

Creates an instances of a pull-style proxy consumers and returns an object reference to the new proxy.

Three varieties of pull-style proxy consumers are defined:

- The [ProxyPullConsumer](#) interface supports connections to pull suppliers that send events as *Anys*.
- The [StructuredProxyPullConsumer](#) interface supports connections to pull suppliers that send structured events.
- The [SequenceProxyPullConsumer](#) interface supports connections to pull suppliers that send sequences of structured events.

The input parameter flag indicates which type of pull style proxy to create.

The target `SupplierAdmin` creates the new pull-style proxy consumer and assigns it a numeric identifier that is unique among all proxy consumers it has created.

### Parameters

<code>ctype</code>	A flag indicating which style of pull-style proxy consumer to create.
<code>proxy_id</code>	The unique identifier of the new proxy consumer.

### Exceptions

[AdminLimitExceeded](#) The number of consumers currently connected to the channel that the target `SupplierAdmin` is associated with exceeds the value of the `MaxSuppliers` administrative property.

## SupplierAdmin::obtain\_notification\_push\_consumer()

```
ProxyConsumer obtain_notification_push_consumer (  
    in ClientType ctype,
```

---

```
    out ProxyID proxy_id)
    raises ( AdminLimitExceeded );
```

Creates an instance of a push-style proxy supplier and returns an object reference to the new proxy.

Three varieties of push-style proxy consumer are defined:

- The [ProxyPushConsumer](#) interface supports connections to push consumers that receive events as `Anys`.
- The [StructuredProxyPushConsumer](#) interface supports connections to push consumers that receive structured events.
- The [SequenceProxyPushConsumer](#) interface supports connections to push consumers that receive sequences of structured events.

The input parameter `flag` indicates which type of push-style proxy to create.

The target `SupplierAdmin` creates the new push-style proxy consumer and assigns it a numeric identifier that is unique among all proxy suppliers it has created.

## Parameters

<code>ctype</code>	A flag that indicates the type of push-style proxy consumer to create.
<code>proxy_id</code>	The unique identifier of the new proxy consumer.

## Exceptions

[AdminLimitExceeded](#) The number of consumers currently connected to the channel that the target `SupplierAdmin` is associated with exceeds the value of the `MaxSuppliers` administrative property.

## SupplierAdmin::destroy()

```
void destroy();
```

Iteratively destroys each proxy under the administration of the target object, and finally destroys the target object itself. When destroying each object, it frees any storage associated with the object, and then invalidates the object's IOR.





# CosNotifyComm Module

CosNotifyComm specifies the following interfaces to instantiate notification service clients:

<a href="#">PushConsumer</a>	<a href="#">PushSupplier</a>
<a href="#">PullConsumer</a>	<a href="#">PullSupplier</a>
<a href="#">StructuredPushConsumer</a>	<a href="#">StructuredPushSupplier</a>
<a href="#">StructuredPullConsumer</a>	<a href="#">StructuredPullSupplier</a>
<a href="#">SequencePushConsumer</a>	<a href="#">SequencePushSupplier</a>
<a href="#">SequencePullConsumer</a>	<a href="#">SequencePullSupplier</a>

The module also specifies the [NotifyPublish](#) and [NotifySubscribe](#) interfaces to facilitate informing notification clients about subscription and publication changes.

## CosNotifyComm Exceptions

### CosNotifyComm::InvalidEventType Exception

```
exception InvalidEventType{ CosNotification::EventType type };
```

Raised when the specified [EventType](#) is not syntactically correct. It returns the name of the invalid event type.

---

**Note:** The Orbix notification service does not throw this exception.

---





# CosNotifyComm::NotifyPublish Interface

```
interface NotifyPublish {
    void offer_change (
        in CosNotification::EventTypeSeq added,
        in CosNotification::EventTypeSeq removed )
        raises ( InvalidEventType );
};
```

The `NotifyPublish` interface supports an operation that allows a supplier to announce, or publish, the names of the event types it supplies. It is an abstract interface which is inherited by all notification service consumer interfaces, and it enables suppliers to inform consumers supporting this interface of the types of events they intend to supply.

## NotifyPublish::offer\_change()

```
void offer_change (
    in CosNotification::EventTypeSeq added,
    in CosNotification::EventTypeSeq removed )
    raises ( InvalidEventType );
```

Allows a supplier of notifications to announce, or publish, the names of the types of events it supplies.

---

**Note:** Each event type name consists of two components: the name of the domain in which the event type has meaning, and the name of the actual event type. Either component of a type name may specify a complete domain/event type name, a domain/event type name containing the wildcard '\*' character, or the special event type name "%ALL".

---

---

## Parameters

added	A sequence of event type names specifying those event types which the event supplier plans to supply.
removed	Sequence of event type names specifying those event types which the client no longer plans to supply.

## Exceptions

<a href="#">InvalidEventType</a>	One of the event type names supplied in either input parameter is syntactically invalid. In this case, the invalid name is returned in the type field of the exception.
----------------------------------	---

# CosNotifyComm::NotifySubscribe Interface

```
interface NotifySubscribe {
    void subscription\_change(
        in CosNotification::EventTypeSeq added,
        in CosNotification::EventTypeSeq removed )
        raises ( InvalidEventType );
};
```

The `NotifySubscribe` interface supports an operation allowing a consumer to inform suppliers of the event types it wishes to receive. It is an abstract interface that is inherited by all notification service supplier interfaces. Its main purpose is to enable consumers to inform suppliers of the event types they are interested in, ultimately enabling the suppliers to avoid supplying events that are not of interest to any consumer.

## `NotifySubscribe::subscription_change()`

```
void subscription_change(
    in CosNotification::EventTypeSeq added,
    in CosNotification::EventTypeSeq removed )
    raises ( InvalidEventType );
```

Allows a consumer to inform suppliers of the event types it wishes to receive.

---

**Note:** Each event type name is comprised of two components: the name of the domain in which the event type has meaning, and the name of the actual event type. Also note that either component of a type name may specify a complete domain/event type name, a domain/event type name containing the wildcard '\*' character, or the special event type name "%ALL".

---

---

## Parameters

added	A sequence of event type names specifying the event types the consumer wants to add to its subscription list.
removed	A sequence of event type names specifying the event types the consumer wants to remove from its subscription list.

## Exceptions

<a href="#">InvalidEventType</a>	One of the event type names supplied in either input parameter is syntactically invalid. The invalid name is returned in the type field of the exception.
----------------------------------	---

# CosNotifyComm::PullConsumer Interface

```
interface PullConsumer :  
    NotifyPublish,  
    CosEventComm::PullConsumer  
{  
};
```

The `PullConsumer` interface inherits all the operations of `CosEventComm::PullConsumer`. In addition, the `PullConsumer` interface inherits the [NotifyPublish](#) interface described above, which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

---

**Note:** An object supporting `PullConsumer` can receive all events that were supplied to its associated channel. How events supplied to the channel in other forms are internally mapped for delivery to a `PullConsumer` is summarized in the *CORBA Notification Service Guide*.

---



# CosNotifyComm::PullSupplier Interface

```
interface PullSupplier :  
    NotifySubscribe,  
    CosEventComm::PullSupplier  
{  
};
```

The `PullSupplier` interface inherits all the operations of `CosEventComm::PullSupplier`. In addition, the `PullSupplier` interface inherits the [NotifySubscribe](#) interface described above, which enables a consumer to inform an instance supporting this interface whenever there is a change to the types of events it wishes to receive.

---

**Note:** An object supporting the `PullSupplier` interface can transmit events that can potentially be received by any consumer connected to the channel. How events supplied to the channel in other forms are translated is summarized in the *CORBA Notification Service Guide*

---





# CosNotifyComm::PushConsumer Interface

```
interface PushConsumer :  
    NotifyPublish,  
    CosEventComm::PushConsumer  
{  
};
```

The `PushConsumer` interface inherits all the operations of `CosEventComm::PushConsumer`. In addition, the `PushConsumer` interface inherits the [NotifyPublish](#) interface described above, which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

---

**Note:** An object supporting `PushConsumer` can receive all events that were supplied to its associated channel. How events supplied to the channel in other forms are internally mapped for delivery to a `PushConsumer` is summarized in the *CORBA Notification Service Guide*.

---



# CosNotifyComm::PushSupplier Interface

```
interface PushSupplier :  
    NotifySubscribe,  
    CosEventComm::PushSupplier  
{  
};
```

The `PushSupplier` interface inherits all the operations of `CosEventComm::PushSupplier`. In addition, the `PushSupplier` interface inherits the [NotifySubscribe](#) interface described above, which enables a consumer to inform an instance supporting this interface whenever there is a change to the types of events it wishes to receive.

---

**Note:** An object supporting the `PushSupplier` interface can transmit events that can potentially be received by any consumer connected to the channel. How events supplied to the channel in other forms are translated is summarized in the *CORBA Notification Service Guide*

---



# CosNotifyComm:: SequencePullConsumer Interface

```
interface SequencePullConsumer : NotifyPublish {  
    void disconnect\_sequence\_pull\_consumer\(\);  
};
```

The `SequencePullConsumer` interface defines an operation to disconnect the pull consumer from its associated supplier. The `SequencePullConsumer` interface inherits [NotifyPublish](#), which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

---

**Note:** An object supporting the `SequencePullConsumer` interface can receive all events that were supplied to its associated channel, including events supplied in a form other than a sequence of structured events. How events supplied to the channel in other forms are internally mapped into a sequence of structured events for delivery to a `SequencePullConsumer` is summarized in the *CORBA Notification Service Guide*.

---

## **SequencePullConsumer:: disconnect\_sequence\_pull\_consumer()**

```
void disconnect_sequence_pull_consumer();
```

Terminates a connection between the target `SequencePullConsumer` and its associated supplier. The target `SequencePullConsumer` releases all resources allocated to support the connection, and disposes of its own object reference.



# CosNotifyComm:: SequencePullSupplier Interface

```
interface SequencePullSupplier : NotifySubscribe
{
    CosNotification::EventBatch pull\_structured\_events(
        in long max_number )
        raises(CosEventComm::Disconnected);

    CosNotification::StructuredEvent try\_pull\_structured\_events(
        in long max_number,
        out boolean has_event)
        raises(CosEventComm::Disconnected);

    void disconnect\_sequence\_pull\_supplier();
};
```

The `SequencePullSupplier` interface supports operations that enable suppliers to transmit sequences of structured events using the pull model. It also defines an operation to disconnect the pull supplier from its associated consumer. The `SequencePullSupplier` interface inherits [NotifySubscribe](#), which enables a consumer to inform an instance supporting this interface whenever there is a change to the types of events it is interested in receiving.

---

**Note:** An object supporting the `SequencePullSupplier` interface can transmit events that can be received by any consumer connected to the channel, including those which consume events in a form other than a sequence of structured events. How events supplied to the channel in the form of a sequence of structured events are internally mapped into different forms for delivery to consumers that receive events in a form other than the a sequence of structured events is summarized in the *CORBA Notification Service Guide*.

---

---

## SequencePullSupplier::pull\_structured\_events()

```
CosNotification::EventBatch pull_structured_events(  
    in long max_number )  
    raises(CosEventComm::Disconnected);
```

Blocks until a sequence of structured events is available for transmission, at which time it returns the sequence containing events to be delivered to its connected consumer proxy.

The amount of time the supplier packs events into the sequence before transmitting it, along with the maximum size of any sequence it transmits (regardless of the input parameter), are controlled by QoS property settings as described in the *CORBA Notification Service Guide*.

### Parameters

`max_number`      The maximum length of the sequence returned.

### Exceptions

`Disconnected`    The operation was invoked on a `SequencePullSupplier` that is not currently connected to a consumer proxy.

## SequencePullSupplier::try\_pull\_structured\_events()

```
CosNotification::StructuredEvent try_pull_structured_events(  
    in long max_number,  
    out boolean has_event)  
    raises(CosEventComm::Disconnected);
```

Returns a sequence of a structured events that contains events being delivered to its connected consumer, if such a sequence is available for delivery at the time the operation was invoked:

- If an event sequence is available for delivery and is returned as the result, the output parameter `has_event` is set to `TRUE`.
- If no event sequence is available to return upon invocation, the operation returns immediately with the value of the output parameter set to `FALSE`. In this case, the return value does not contain a valid event sequence.



---

## Parameters

<code>max_number</code>	The maximum length of the sequence returned.
<code>has_event</code>	An output parameter of type <code>boolean</code> that indicates whether or not the return value actually contains a sequence of events.

## Exceptions

<code>Disconnected</code>	This operation was invoked on a <code>SequencePullSupplier</code> that is not currently connected to a consumer proxy.
---------------------------	--

## **`SequencePullSupplier::disconnect_sequence_pull_supplier()`**

```
void disconnect_sequence_pull_supplier();
```

Terminates a connection between the target `SequencePullSupplier` and its associated consumer. The target `SequencePullSupplier` releases all resources allocated to support the connection, and disposes of its own object reference.



# CosNotifyComm::SequencePushConsumer Interface

```
interface SequencePushConsumer : NotifyPublish {  
    void push\_structured\_events(  
        in CosNotification::EventBatch notifications)  
        raises(CosEventComm::Disconnected);  
    void disconnect\_sequence\_push\_consumer\(\);  
};
```

The `SequencePushConsumer` interface supports an operation that enables consumers to receive sequences of structured events using the push model. It also defines an operation to disconnect the push consumer from its associated supplier. The `SequencePushConsumer` interface inherits [NotifyPublish](#), which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

---

**Note:** An object supporting the `SequencePushConsumer` interface can receive all events which are supplied to its associated channel, including events supplied in a form other than a sequence of structured events. How events supplied to the channel in other forms are internally mapped into a sequence of structured events for delivery to a `SequencePushConsumer` is summarized in the *CORBA Notification Service Guide*.

---

## SequencePushConsumer::push\_structured\_events()

```
void push_structured_events(  
    in CosNotification::EventBatch notifications)  
    raises(CosEventComm::Disconnected);
```

Enables consumers to receive sequences of structured events by the push model.

---

The maximum number of events that are transmitted within a single invocation of this operation, along with the amount of time a supplier of sequences of structured events packs individual events into the sequence before invoking this operation, are controlled by QoS property settings as described in the *CORBA Notification Service Guide*.

## Parameters

`notifications` A parameter of type [EventBatch](#) as defined in the [CosNotification](#) module. Upon invocation, this parameter contains a sequence of structured events being delivered to the consumer by its associated supplier proxy.

## Exceptions

`Disconnected` The operation was invoked on a `SequencePushConsumer` instance that is not currently connected to a supplier proxy.

## **SequencePushConsumer:: disconnect\_sequence\_push\_consumer()**

```
void disconnect_sequence_push_consumer();
```

Terminates a connection between the target `SequencePushConsumer` and its associated supplier proxy. The target `SequencePushConsumer` releases all resources allocated to support the connection, and disposes of its own object reference.

# CosNotifyComm:: SequencePushSupplier Interface

```
interface SequencePushSupplier : NotifySubscribe
{
    void disconnect\_sequence\_push\_supplier\(\);
};
```

The `SequencePushSupplier` interface defines an operation that to disconnect the push supplier from its associated consumer proxy. In addition, the `SequencePushSupplier` interface inherits [NotifySubscribe](#), which enables a consumer to inform an instance supporting this interface whenever there is a change to the types of events it is interested in receiving.

---

**Note:** An object supporting the `SequencePushSupplier` interface can transmit events that can be received by any consumer connected to the channel, including those which consume events in a form other than a sequence of structured events. How events supplied to the channel in the form of a sequence of structured events are internally mapped into different forms for delivery to consumers which receive events in a form other than a sequence of structured events is summarized in the *CORBA Notification Service Guide*.

---

## **SequencePushSupplier::disconnect\_sequence\_push\_supplier()**

```
void disconnect_sequence_push_supplier();
```

Terminates a connection between the target `SequencePushSupplier` and its associated consumer. The target `SequencePushSupplier` releases all resources allocated to support the connection, and disposes of its own object reference.



# CosNotifyComm:: StructuredPullConsumer Interface

```
interface StructuredPullConsumer : NotifyPublish
{
    void disconnect\_structured\_pull\_consumer\(\);
};
```

The `StructuredPullConsumer` defines an operation that can be invoked to disconnect the pull consumer from its associated supplier. In addition, the `StructuredPullConsumer` interface inherits the `NotifyPublish` interface, which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

---

**Note:** An object supporting the `StructuredPullConsumer` interface can receive all events that were supplied to its associated channel, including events supplied in a form other than a structured event. How events supplied to the channel in other forms are internally mapped into a structured event for delivery to a `StructuredPullConsumer` is summarized in the *CORBA Notification Service Guide*.

---

## **StructuredPullConsumer:: disconnect\_structured\_pull\_consumer()**

```
void disconnect_structured_pull_consumer();
```

Terminates a connection between the target `StructuredPullConsumer`, and its associated supplier proxy. The target `StructuredPullConsumer` releases all resources allocated to support the connection, and disposes of its own object reference.





# CosNotifyComm:: StructuredPullSupplier Interface

```
interface StructuredPullSupplier : NotifySubscribe
{
    CosNotification::StructuredEvent pull_structured_event()
        raises(CosEventComm::Disconnected);

    CosNotification::StructuredEvent try_pull_structured_event(
        out boolean has_event)
        raises(CosEventComm::Disconnected);

    void disconnect\_structured\_pull\_supplier\(\);
};
```

The `StructuredPullSupplier` interface supports operations that enable suppliers to transmit structured events by the pull model. It also defines an operation to disconnect the pull supplier from its associated consumer proxy. In addition, the `StructuredPullSupplier` interface inherits the [NotifySubscribe](#) interface, which enables a consumer to inform an instance supporting this interface whenever there is a change to the types of events it is interested in receiving.

---

**Note:** An object supporting the `StructuredPullSupplier` interface can transmit events that can potentially be received by any consumer connected to the channel, including those which consume events in a form other than a structured event. How events supplied to the channel in other forms are translated is summarized in the *CORBA Notification Service Guide*

---

## StructuredPullSupplier::pull\_structured\_event()

```
CosNotification::StructuredEvent pull_structured_event()
    raises(CosEventComm::Disconnected);
```

---

Blocks until an event is available for transmission, at which time it returns an instance of a structured event containing the event being delivered to its connected consumer proxy.

## Exceptions

`Disconnected` The operation was invoked on a `StructuredPullSupplier` that is not currently connected to a consumer proxy.

## **StructuredPullSupplier::try\_pull\_structured\_event()**

```
CosNotification::StructuredEvent try_pull_structured_event(  
    out boolean has_event)  
    raises(CosEventComm::Disconnected);
```

If an event is available for delivery at the time the operation was invoked, the method returns a structured event that contains the event being delivered to its connected consumer and the output parameter of the operation is set to `TRUE`. If no event is available to return upon invocation, the operation returns immediately with the value of the output parameter set to `FALSE`. In this case, the return value does not contain a valid event.

## Parameters

`has_event` An output parameter of type `boolean` that indicates whether or not the return value actually contains an event.

## Exceptions

`Disconnected` The operation was invoked on a `StructuredPullSupplier` that is not currently connected to a consumer proxy.

## **StructuredPullSupplier::disconnect\_structured\_pull\_supplier()**

```
void disconnect_structured_pull_supplier();
```

Terminates a connection between the target `StructuredPullSupplier` and its associated consumer. The target `StructuredPullSupplier` releases all resources allocated to support the connection, and disposes of its own object reference.

# CosNotifyComm:: StructuredPushConsumer Interface

```
interface StructuredPushConsumer : NotifyPublish {  
    void push\_structured\_event(  
        in CosNotification::StructuredEvent notification)  
        raises(CosEventComm::Disconnected);  
    void disconnect\_structured\_push\_consumer();  
};
```

The `StructuredPushConsumer` interface supports an operation enabling consumers to receive structured events by the push model. It also defines an operation to disconnect the push consumer from its associated proxy supplier. In addition, the `StructuredPushConsumer` interface inherits the [NotifyPublish](#) interface described above, which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

---

**Note:** An object supporting the `StructuredPushConsumer` interface can receive all events that were supplied to its associated channel, including events supplied in a form other than a structured event. How events supplied to the channel in other forms are internally mapped into a structured event for delivery to a `StructuredPushConsumer` is summarized in the *CORBA Notification Service Guide*.

---

## StructuredPushConsumer::push\_structured\_event()

```
void push_structured_event(  
    in CosNotification::StructuredEvent notification)  
    raises(CosEventComm::Disconnected);
```

Enables consumers to receive structured events by the push model.

---

## Parameters

`notification` A parameter of type [StructuredEvent](#) as defined in the [CosNotification](#) module. When the method returns this parameter contains a structured event being delivered to the consumer by its proxy supplier.

## Exceptions

`Disconnected` This operation was invoked on a `StructuredPushConsumer` instance that is not currently connected to a proxy supplier.

### **StructuredPushConsumer:: disconnect\_structured\_push\_consumer()**

```
void disconnect_structured_push_consumer();
```

Terminates a connection between the target `StructuredPushConsumer` and its associated proxy supplier. That the target `StructuredPushConsumer` releases all resources allocated to support the connection, and disposes of its own object reference.

# CosNotifyComm:: StructuredPushSupplier Interface

```
interface StructuredPushSupplier : NotifySubscribe {  
    void disconnect\_structured\_push\_supplier\(\);  
};
```

The `StructuredPushSupplier` interface supports the behavior of objects that transmit structured events using push-style communication. It defines an operation that can be invoked to disconnect the push supplier from its associated consumer proxy. In addition, the `StructuredPushSupplier` interface inherits `NotifySubscribe`, which enables a consumer to inform an instance supporting this interface whenever there is a change to the types of events it is interested in receiving.

---

**Note:** An object supporting the `StructuredPushSupplier` interface can transmit events which can potentially be received by any consumer connected to the channel, including those which consume events in a form other than a structured event. How events supplied to the channel are translated is summarized in the *CORBA Notification Service Guide*.

---

## **StructuredPushSupplier:: disconnect\_structured\_push\_supplier()**

```
void disconnect_structured_push_supplier();
```

Terminates a connection between the target `StructuredPushSupplier`, and its associated consumer. The target `StructuredPushSupplier` releases all resources allocated to support the connection, and disposes of its own object reference.



# CosNotifyFilter Module

The `CosNotifyFilterModule` specifies the following interfaces to support event filtering:

[Filter](#)  
[FilterFactory](#)  
[MappingFilter](#)  
[FilterAdmin](#)

In addition to these interfaces the module specifies several data types and exceptions related to event filtering.

## CosNotifyFilter Data Types

### CosNotifyFilter::ConstraintID Data Type

```
typedef long ConstraintID;
```

Identifies a constraint.

### CosNotifyFilter::ConstraintExp Data Structure

```
struct ConstraintExp
{
    CosNotification::EventTypeSeq event_types;
    string constraint_expr;
};
```

Contains a constraint expression and a list of events to check against. The `constraint_expr` member is a string that conforms to the Trader constraint grammar. For more information on the constraint grammar, see the *CORBA Notification Service Guide*.

---

## CosNotifyFilter::ContsrainIDSeq Data Type

```
typedef <ConstraintID> ConstraintIDSeq;
```

Contains a list of constraint ID.

## CosNotifyFilter::ConstraintExpSeq Data Type

```
typedef sequence<ConstraintExp> ContsrainExpSeq;
```

Contains a list of constraint expressions.

## CosNotifyFilter::ConstraintInfo Data Structure

```
struct ConstraintInfo
{
    ConstraintExp constraint_expression;
    ConstraintID constraint_id;
}
```

Specifies an instantiated constraint.

## CosNotifyFilter::ConstraintInfoSeq Data Type

```
typedef sequence<ConstraintInfo> ConstraintInfoSeq;
```

Contains a list of instantiated constraints.

## CosNotifyFilter::FilterID Data Type

```
typedef long FilterID;
```

Identifies an instantiated filter. It is unique to the object to which it is attached.

## CosNotifyFilter::FilterIDSeq Data Type

```
typedef sequence<FilterID> FilterIDSeq;
```



Contains a list of FilterIds.

### **CosNotifyFilter::MappingConstraintPair Data Structure**

```
struct MappingConstraintPair
{
    ConstraintExp constraint_expression;
    any          result_to_set;
}
```

Specifies a constraint expression and the value to set if the event matches the constraint expression.

### **CosNotifyFilter::MappingConstraintPairSeq Data Type**

```
typedef sequence<MappingConstraintPair> MappingConstraintPairSeq
```

Contains a list of mapping filter constraint/value pairs.

### **CosNotifyFilter::MappingConstraintInfo Data Structure**

```
struct MappingConstraintInfo
{
    ConstraintExp constraint_expression;
    ConstraintID  constraint_id;
    any          value;
}
```

Specifies a mapping constraint that has been instantiated.

### **CosNotifyFilter::MappingConstraintInfoSeq Data Types**

```
typedef sequence<MappingConstraintInfo> MappingConstraintInfoSeq;
```

Contains a list of instantiated mapping filter constraint/value pairs.

---

## CosNotifyFilter::CallbackID Data Type

```
typedef long CallbackID;
```

Holds an identifier for a callback registered with [attach\\_callback](#).

## CosNotifyFilter::CallbackIDSeq Data Type

```
typedef sequence<CallbackID> CallbackIDSeq;
```

Contains a list of callback IDs.

# CosNotifyFilter Exceptions

## CosNotifyFilter::UnsupportedFilterableData Exception

```
exception UnsupportedFilterableData {};
```

Raised if the input parameter contains data that the `match` operation is not designed to handle. For example, the filterable data contains a field whose name corresponds to a standard event field that has a numeric value, but the actual value associated with this field name within the event is a string.

## CosNotifyFilter::InvalidGrammar Exception

```
exception InvalidGrammar {};
```

Raised when creating a filter. If the string passed to the filter factory specifies a grammar that is not supported, the factory will throw `InvalidGrammar`.

---

**Note:** Orbix notification service supports the `EXTENDED_TCL` grammar.

---

### **CosNotifyFilter::InvalidConstraint Exception**

```
exception InvalidConstraint {ConstraintExp constr};
```

Raised during the creation of constraints. If the string specifying the constraint is syntactically incorrect, `InvalidConstraint` is thrown. It returns the invalid constraint.

### **CosNotifyFilter::ConstraintNotFound Exception**

```
exception ConstraintNotFound {ConstraintID id};
```

Raised when a specified constraint ID cannot be resolved to a constraint attached to the target filter object. It returns the ID that cannot be resolved.

### **CosNotifyFilterFilter::CallbackNotFound Exception**

```
exception CallbackNotFound {};
```

Raised when the specified callback ID cannot be resolved to a callback object attached to the target filter object.

### **CosNotifyFilter::InvalidValue Exception**

```
exception InvalidValue {ConstraintExp constr; any value};
```

Raised when the `type_code` of the value associated with the mapping filter constraint does not match the [value\\_type](#) of the target mapping filter object.

### **CosNotifyFilter::FilterNotFound Exception**

```
exception FilterNotFound {};
```

Raised if the specified filter ID cannot be resolved to a filter associated with the target object.



# CosNotifyFilter::Filter Interface

```
interface Filter
{
    readonly attribute string constraint\_grammar;

    ConstraintInfoSeq add\_constraints(
        in ConstraintExpSeq constraint_list)
    raises (InvalidConstraint);

    void modify\_constraints(
        in ConstraintIDSeq del_list,
        in ConstraintInfoSeq modify_list)
    raises (InvalidConstraint, ConstraintNotFound);

    ConstraintInfoSeq get\_constraints(
        in ConstraintIDSeq id_list)
    raises (ConstraintNotFound);

    ConstraintInfoSeq get\_all\_constraints();

    void remove\_all\_constraints();

    void destroy();

    boolean match( in any filterable_data )
    raises (UnsupportedFilterableData);

    boolean match\_structured(
        in CosNotification::StructuredEvent filterable_data )
    raises (UnsupportedFilterableData);

    boolean match_typed (
        in CosTrading::PropertySeq filterable_data )
    raises (UnsupportedFilterableData);

    CallbackID attach\_callback (
        in CosNotifyComm::NotifySubscribe callback);
}
```

---

```
void detach_callback ( in CallbackID callback)
raises (CallbackNotFound);

    CallbackIDSeq get_callbacks();
}; // Filter
```

The `Filter` interface defines the behaviors supported by filter objects. These objects encapsulate constraints that are used by the proxies and admins associated with an event channel. The proxies and admins use the constraint definitions to determine which events are forwarded, and which are discarded.

For more information on filters and the constraint language, see the *CORBA Notification Service Guide*.

The `Filter` interface supports operations to manage the constraints associated with a `Filter` instance, along with a read-only attribute to identify the constraint grammar used to evaluate the constraints associated with the instance. In addition, the `Filter` interface supports three variants of the `match` operation which are invoked by a proxy object upon receipt of an event—the specific variant selected depends upon whether the event is received as an `Any` or a structured event—to evaluate the object using the constraints associated with the filter object.

The `Filter` interface also supports operations enabling a client to associate any number of callbacks with the target filter object. The callbacks are notified each time there is a change to the list of event types the filter forwards through the event channel. Operations are also defined to support administration of this callback list by unique identifier.

### **Filter::constraint\_grammar**

```
readonly attribute string constraint_grammar;
```

`constraint_grammar` is a readonly attribute specifying the particular grammar used to parse the constraint expressions encapsulated by the target filter. The value of this attribute is set upon creation of a filter object.

A filter's constraints must be expressed using a particular constraint grammar because its member `match` operations must be able to parse the constraints to determine whether or not a particular event satisfies one of them.

---

Orbix supports an implementation of the `Filter` interface which supports the default constraint grammar described in the *CORBA Notification Service Guide*. The `constraint_grammar` attribute is set to the value `EXTENDED_TCL` when the target filter object supports this default grammar.

Other implementations can provide additional implementations of the `Filter` interface that support different constraint grammars, and thus the `constraint_grammar` attribute must be set to a different value upon creation of such a filter object.

### **Filter::add\_constraints()**

```
ConstraintInfoSeq add_constraints(  
    in ConstraintExpSeq constraint_list)  
    raises (InvalidConstraint);
```

Associates one or more new constraints with the target filter object. Upon successful processing of all input constraint expressions, `add_constraints()` returns a [ConstraintInfoSeq](#) containing all of the constraints and the identifiers assigned to them by the filter.

If one or more of the constraints passed into `add_constraints()` is invalid, none of the constraints are added to the target filter.

---

**Note:** Once `add_constraints()` is invoked by a client, the target filter is temporarily disabled from usage by any proxy or admin it may be associated with. Upon completion of the operation, the target filter is re-enabled and can once again be used by associated proxies and admins to make event forwarding decisions.

---

#### **Parameters**

<code>constraint_list</code>	A sequence of constraint data structures using the constraint grammar supported by the target object.
------------------------------	---

#### **Exceptions**

If any of the constraints in the input sequence is not a valid expression within the supported constraint grammar, the [InvalidConstraint](#) exception is raised. This exception contains as data the specific constraint expression that was determined to be invalid.

---

## Filter::modify\_constraints()

```
void modify_constraints (  
    in ConstraintIDSeq del_list,  
    in ConstraintInfoSeq modify_list)  
    raises (InvalidConstraint, ConstraintNotFound);
```

Modifies the constraints associated with the target filter object. This operation can be used both to remove constraints currently associated with the target filter, and to modify the constraint expressions of constraints currently associated with the filter.

If an exception is raised during the operation, no changes are made to the filter's constraints.

---

**Note:** Once `modify_constraints` is invoked by a client, the target filter is temporarily disabled from use by any proxy or admin. Upon completion of the operation, the target filter is re-enabled and can once again be used by associated proxies and admins to make event forwarding decisions.

---

### Parameters

<code>del_list</code>	A sequence of numeric identifiers each of which should be associated with one of the constraints currently encapsulated by the target filter object.
<code>modify_list</code>	A sequence containing constraint structures and an associated numeric value. The numeric value in each element of the sequence is the unique identifier of one of the constraints encapsulated by the target filter.

### Exceptions

[ConstraintNotFound](#) Raised if any of the numeric values in either input sequences does not correspond to the unique identifier associated with any constraint encapsulated by the target filter. This exception contains the specific identifier that did not correspond to the identifier of some constraint encapsulated by the target filter.



---

[InvalidConstraint](#) Raised if any of the constraint expressions supplied in the second input sequence is not a valid expression in terms of the constraint grammar supported by the target object. This exception contains the specific constraint that was determined to be invalid.

### **Filter::get\_constraints()**

[ConstraintInfoSeq](#) get\_constraints(in [ConstraintIDSeq](#) id\_list)  
raises ([ConstraintNotFound](#));

Returns a sequence of data structures containing the input identifiers along with their associated constraint.

#### **Parameters**

id\_list A sequence of numeric values corresponding to the unique identifiers of constraints encapsulated by the target object.

#### **Exceptions**

[ConstraintNotFound](#) One of the input values does not correspond to the identifier of some encapsulated constraint. The exception contains that input value.

### **Filter::get\_all\_constraints()**

[ConstraintInfoSeq](#) get\_all\_constraints();

Returns all of the constraints currently encapsulated by the target filter object.

### **Filter::remove\_all\_constraints()**

void remove\_all\_constraints();

Removes all of the constraints currently encapsulated by the target filter. Upon completion, the target filter still exists but no constraints are associated with it.

---

## Filter::destroy()

```
void destroy();
```

Destroys the target filter and invalidates its object reference.

## Filter::match()

```
boolean match (in any filterable_data)  
    raises (UnsupportedFilterableData);
```

Evaluates the filter constraints associated with the target filter against an event supplied to the channel in the form of a `CORBA::Any`. The operation returns `TRUE` if the input event satisfies one of the filter constraints, and `FALSE` otherwise.

The act of determining whether or not a given event passes a given filter constraint is specific to the type of grammar in which the filter constraint is specified.

### Parameters

`filterable_data` A `CORBA::Any` which contains an event to be evaluated.

### Exceptions

[UnsupportedFilterableData](#) The input parameter contains data that the `match` operation is not designed to handle.

## Filter::match\_structured()

```
boolean match_structured(  
    in CosNotification::StructuredEvent filterable_data )  
    raises (UnsupportedFilterableData);
```

Evaluates the filter constraints associated with the target filter against a structured event. The operation returns `TRUE` if the input event satisfies one of the filter constraints, and `FALSE` otherwise.

The act of determining whether or not a given event passes a given filter constraint is specific to the type of grammar in which the filter constraint is specified.

---

## Parameters

`filterable_data` A [CosNotification::StructuredEvent](#) containing an event to be evaluated,

## Exceptions

[UnsupportedFilterableData](#) The input parameter contains data that the `match` operation is not designed to handle.

## Filter::attach\_callback()

```
CallbackID attach_callback (  
    in CosNotifyComm::NotifySubscribe callback);
```

Associates an object supporting the [CosNotifyComm::NotifySubscribe](#) interface with the target filter. This operation returns a numeric value assigned to this callback that is unique to all such callbacks currently associated with the target filter.

After this operation has been successfully invoked on a filter, the filter invokes the [subscription\\_change\(\)](#) method of all its associated callbacks each time the set of constraints associated with the filter is modified. This process informs suppliers in the filter's callback list of the change in the set of event types to which the filter's clients subscribe. With this information, suppliers can make intelligent decisions about which event types to produce.

## Parameters

`callback` The reference to an object supporting the [CosNotifyComm::NotifySubscribe](#) interface.

## Filter::detach\_callback()

```
void detach_callback(in CallbackID callback)  
raises (CallbackNotFound);
```

Removes a callback object from the filter's callback list. Subsequent changes to the event type subscription list encapsulated by the target filter are no longer propagated to that callback object.

---

## Parameters

`callback` A unique identifiers associated with one of the callback objects attached to the target filter.

## Exceptions

[CallbackNotFound](#) The input value does not correspond to the unique identifier of a callback object currently attached to the target filter object.

## **Filter::get\_callbacks()**

[CallbackIDSeq](#) `get_callbacks()`;

Returns all the unique identifiers for the callback objects attached to the target filter.

# CosNotifyFilter::FilterAdmin Interface

```
interface FilterAdmin {  
    FilterID add\_filter ( in Filter new_filter );  
  
    void remove\_filter ( in FilterID filter )  
        raises ( FilterNotFound );  
  
    Filter get\_filter ( in FilterID filter )  
        raises ( FilterNotFound );  
  
    FilterIDSeq get\_all\_filters();  
  
    void remove\_all\_filters();  
};
```

The `FilterAdmin` interface defines operations enabling an object supporting this interface to manage a list of filters, each of which supports the [Filter](#) interface. This interface is an abstract interface which is inherited by all of the proxy and admin interfaces defined by the notification service.

## FilterAdmin::add\_filter()

```
FilterID add_filter(in Filter new_filter);
```

Appends a filter to the list of filters associated with the target object upon which the operation was invoked and returns an identifier for the filter.

### Parameters

`new_filter`      A reference to an object supporting the [Filter](#) interface.

## FilterAdmin::remove\_filter()

```
void remove_filter(in FilterID filter)  
    raises ( FilterNotFound );
```

---

Removes the specified filter from the target object's list of filters.

### Parameters

`filter` A numeric value identifying a filter associated with the target object

### Exceptions

[FilterNotFound](#) The identifier does not correspond to a filter associated with the target object.

## FilterAdmin::get\_filter()

```
Filter get_filter (in FilterID filter)  
    raises ( FilterNotFound );
```

Returns the object reference to the specified filter.

### Parameters

`filter` A numeric value identifying a filter associated with the target object

### Exceptions

[FilterNotFound](#) The identifier does not correspond to a filter associated with the target object.

## FilterAdmin::get\_all\_filters()

```
FilterIDSeq get_all_filters();
```

Returns the list of unique identifiers corresponding to all of the filters associated with the target object.

## FilterAdmin::remove\_all\_filters()

```
void remove_all_filters();
```

---

Removes all filters from the filter list of the target object.





# CosNotifyFilter::FilterFactory Interface

```
interface FilterFactory {
    Filter create\_filter (
        in string constraint_grammar)
        raises (InvalidGrammar);

    MappingFilter create\_mapping\_filter (
        in string constraint_grammar,
        in any default_value)
        raises(InvalidGrammar);
};
```

The `FilterFactory` interface defines operations for creating filter.

## **FilterFactory::create\_filter()**

```
Filter create\_filter (in string constraint_grammar)
    raises (InvalidGrammar);
```

Creates a forwarding filter object and returns a reference to the new filter.

### **Parameters**

`constraint_grammar` A string identifying the grammar used to parse constraints associated with this filter.

### **Exceptions**

[InvalidGrammar](#) The client invoking this operation supplied the name of a grammar that is not supported by any forwarding filter implementation this factory is capable of creating.

---

## FilterFactory::create\_mapping\_filter()

```
MappingFilter create_mapping_filter (  
    in string constraint_grammar,  
    in any default_value)  
    raises(InvalidGrammar);
```

Creates a mapping filter object and returns a reference to the new mapping filter.

### Parameters

`constraint_grammar` A string parameter identifying the grammar used to parse constraints associated with this filter.

`default_value` An `Any` specifying the `default_value` of the new mapping filter.

### Exceptions

[InvalidGrammar](#) The client invoking this operation supplied the name of a grammar that is not supported by any mapping filter implementation this factory is capable of creating.

# CosNotifyFilter::MappingFilter Interface

```
interface MappingFilter
{
    readonly attribute string constraint\_grammar;
    readonly attribute CORBA::TypeCode value\_type;
    readonly attribute any default\_value;

    MappingConstraintInfoSeq add\_mapping\_constraints (
        in MappingConstraintPairSeq pair_list)
    raises (InvalidConstraint, InvalidValue);

    void modify\_mapping\_constraints (
        in ConstraintIDSeq del_list,
        in MappingConstraintInfoSeq modify_list)
    raises (InvalidConstraint, InvalidValue, ConstraintNotFound);

    MappingConstraintInfoSeq get\_mapping\_constraints (
        in ConstraintIDSeq id_list)
    raises (ConstraintNotFound);

    MappingConstraintInfoSeq get\_all\_mapping\_constraints();

    void remove\_all\_mapping\_constraints();

    void destroy();

    boolean match ( in any filterable_data, out any result_to_set )
    raises (UnsupportedFilterableData);

    boolean match\_structured (
        in CosNotification::StructuredEvent filterable_data,
        out any result_to_set)
    raises (UnsupportedFilterableData);

    boolean match\_typed (
```

---

```
        in CosTrading::PropertySeq filterable_data,
        out any result_to_set)
    raises (UnsupportedFilterableData);
}; // MappingFilter
```

The `MappingFilter` interface defines the behaviors of objects that encapsulate a sequence of constraint-value pairs (see the description of the Default Filter Constraint Language in the *CORBA Notification Service Guide*). These constraint-value pairs are used to evaluate events and adjust their lifetime/priority values according to the result. An object supporting the `MappingFilter` interface can effect either an events lifetime property or its priority property, but not both.

The `MappingFilter` interface supports the operations required to manage the constraint-value pairs associated with an object instance supporting the interface. In addition, the `MappingFilter` interface supports a read-only attribute that identifies the constraint grammar used to parse the constraints encapsulated by this object. The `MappingFilter` interface supports a read-only attribute that identifies the typecode associated with the datatype of the specific property value it is intended to affect. It also supports another read-only attribute which holds the default value which is returned as the result of a match operation in cases when the event in question is found to satisfy none of the constraints encapsulated by the mapping filter. Lastly, the `MappingFilter` interface supports three variants of the operation which are invoked by an associated proxy object upon receipt of an event, to determine how the property of the event which the target mapping filter object was designed to affect should be modified.

### **MappingFilter::constraint\_grammar**

```
readonly attribute string constraint_grammar;
```

Identifies the grammar used to parse the constraint expressions encapsulated by the target mapping filter. The value of this attribute is set upon creation of a mapping filter.

A filter object's constraints must be expressed using a particular constraint grammar because its member `match` operations must be able to parse the constraints to determine whether or not a particular event satisfies one of them.

---

Orbix supports an implementation of the `MappingFilter` object which supports the default constraint grammar described in the *CORBA Notification Service Guide*. `constraint_grammar` is set to the value `EXTENDED_TCL` when the target mapping filter supports this default grammar.

Users may provide additional implementations of the `MappingFilter` interface which support different constraint grammars, and thus set the `constraint_grammar` attribute to a different value when creating such a mapping filter.

### **MappingFilter::value\_type**

readonly attribute `CORBA::TypeCode` `value_type`;

Identifies the datatype of the property value that the target mapping filter is designed to affect. Note that the factory creation operation for mapping filters accepts as an input parameter the `default_value` to associate with the mapping filter instance. This `default_value` is a `CORBA::Any`. Upon creation of a mapping filter, the `typecode` associated with the `default_value` is abstracted from the `CORBA::Any`, and its value is assigned to this attribute.

### **MappingFilter::default\_value**

readonly attribute `any` `default_value`;

The value returned as the result of any `match` operation during which the input event does not satisfy any of the constraints encapsulated by the mapping filter. The value of this attribute is set upon creation of a mapping filter object instance.

### **MappingFilter::add\_mapping\_constraints()**

```
MappingConstraintInfoSeq add_mapping_constraints (  
    in MappingConstraintPairSeq pair_list)  
    raises (InvalidConstraint, InvalidValue);
```

Returns a sequence of structures which contain one of the input constraint expressions, its corresponding value, and the unique identifier assigned to this constraint-value pair by the target filter.

---

If one or more of the constraints passed into `add_mapping_constraints()` is invalid, none of the constraints are added to the target mapping filter.

---

**Note:** Once `add_mapping_constraints` is invoked by a client, the target filter is temporarily disabled from use by any proxy it may be associated with. Upon completion of the operation, the target filter is re-enabled and can once again be used by associated proxies to make event property mapping decisions.

---

## Parameters

`pair_list` A sequence of constraint-value pairs. Each constraint in this sequence must be expressed in the constraint grammar supported by the target object, and each associated value must be of the data type indicated by the [value\\_type](#) attribute of the target object.

## Exceptions

[InvalidConstraint](#) Raised if any of the constraint expressions in the input sequence is not a valid expression. This exception contains the constraint that was determined to be invalid.

[InvalidValue](#) Raised if any of the values supplied in the input sequence are not of the same datatype as that indicated by the target object's [value\\_type](#) attribute. This exception contains the invalid value and its corresponding constraint.

## MappingFilter::modify\_mapping\_constraints()

```
void modify_mapping_constraints (
    in ConstraintIDSeq del_list,
    in MappingConstraintInfoSeq modify_list)
    raises(InvalidConstraint,
          InvalidValue,
          ConstraintNotFound);
```

---

Modifies the constraint-value pairs associated with the target mapping filter. This operation can remove constraint-value pairs currently associated with the target mapping filter, and to modify the constraints and/or values of constraint-value pairs currently associated with the target mapping filter.

If an exception is raised during the operation, no changes are made to the filter's constraints.

---

**Note:** Once `modify_mapping_constraints()` is invoked by a client, the target mapping filter is temporarily disabled from use by any proxy it may be associated with. Upon completion of the operation, the target mapping filter is re-enabled and can be used by associated proxies to make event property mapping decisions.

---

## Parameters

<code>del_list</code>	A sequence of unique identifiers associated with one of the constraint-value pairs currently encapsulated by the target mapping filter. If all input values are valid, the specific constraint-value pairs identified by the values contained in this parameter are deleted from the mapping filter's list of constraint-value-pairs.
<code>modify_list</code>	A sequence of structures containing a constraint structure, an <code>Any</code> value, and a numeric identifier. The numeric identifier of each element is the unique identifier associated with one of the constraint-value pairs currently encapsulated by the target filter object. The constraint-value pairs identified are modified to the values specified in the input list.

## Exceptions

[ConstraintNotFound](#) Raised if any of the identifiers in either of the input sequences does not correspond to the unique identifier associated with a constraint-value pair encapsulated by the target mapping filter. This exception contains the identifier which did not correspond to a constraint-value pair encapsulated by the target object.

---

[InvalidConstraint](#) Raised if any of the constraint expressions supplied in an element of the second input sequence is not valid. This exception contains the constraint that was determined to be invalid.

[InvalidValue](#) Raised if any of the values in the second input sequence is not of the same datatype as that indicated by the mapping filter's [value\\_type](#) attribute. This exception contains the invalid value and its corresponding constraint expression.

## MappingFilter::get\_mapping\_constraints()

```
MappingConstraintInfoSeq get_mapping_constraints (  
    in ConstraintIDSeq id_list)  
    raises (ConstraintNotFound);
```

Returns a sequence of constraint-value pairs associated with the target mapping filter.

### Parameters

`id_list` A sequence of unique identifiers for constraint-value pairs encapsulated by the target object.

### Exceptions

[ConstraintNotFound](#) One of the input values does not correspond to the identifier of an encapsulated constraint-value pair. The exception contains the identifier that did not correspond to a constraint-value pair.

## MappingFilter::get\_all\_mapping\_constraints()

```
MappingConstraintInfoSeq get_all_mapping_constraints();
```

Returns all of the constraint-value pairs encapsulated by the target mapping filter.



---

## MappingFilter::remove\_all\_mapping\_constraints

```
void remove_all_mapping_constraints();
```

Removes all of the constraint-value pairs currently encapsulated by the target mapping filter. Upon completion, the target mapping filter still exists but has no constraint-value pairs associated with it.

## MappingFilter::destroy()

```
void destroy();
```

Destroys the target mapping filter, and invalidates its object reference.

## MappingFilter::match()

```
boolean match(in any filterable_data, out any result_to_set)  
    raises (UnsupportedFilterableData);
```

Determines how to modify some property value of an event in the form of a CORBA::Any.

The target mapping filter begins applying the its constraints according to each constraint's associated value, starting with the constraint with the best associated value for the specific property the mapping filter is designed to affect (for example, the highest priority, the longest lifetime, and so on), and ending with the constraint with the worst associated value.

Upon encountering a constraint which the event matches, the operation sets `result_to_set` to the value associated with the matched constraint, and returns with a value of `TRUE`. If the event does not satisfy any of the target mapping filter's constraints, the operation sets `result_to_set` to the value of the target mapping filter's [default\\_value](#) attribute and returns with a value of `FALSE`.

The act of determining whether or not a given event passes a given filter constraint is specific to the grammar used to parse the filter constraints.

### Parameters

`filterable_data` An `Any` containing the event being evaluated

---

`result_to_set` An `Any` containing the value and the property name to set when an event evaluates to `TRUE`.

## Exceptions

[UnsupportedFilterableData](#) The input parameter contains data that the `match` operation is not designed to handle.

## MappingFilter::match\_structured()

```
boolean match_structured (  
    in CosNotification::StructuredEvent filterable_data,  
    out any result_to_set)  
    raises (UnsupportedFilterableData);
```

Determines how to modify some property value of a structured event.

The target mapping filter begins applying the its constraints according to each constraints associated value, starting with the constraint with the best associated value for the specific property the mapping filter is designed to affect (for example, the highest priority, the longest lifetime, and so on), and ending with the constraint with the worst associated value.

Upon encountering a constraint which the event matches, the operation sets `result_to_set` to the value associated with the matched constraint, and returns with a value of `TRUE`. If the event does not satisfy any of the target mapping filter's constraints, the operation sets `result_to_set` to the value of the target mapping filter's [default\\_value](#) attribute and returns with a value of `FALSE`.

The act of determining whether or not a given event passes a given filter constraint is specific to the grammar used to parse the filter constraints.

## Parameters

`filterable_data` A [CosNotification::StructuredEvent](#) containing the event being evaluated.

`result_to_set` An `Any` containing the value and the property name to set when an event evaluates to `TRUE`.

---

## Exceptions

[UnsupportedFilterableDat](#)The input parameter contains data that `match_structured()` is not designed to handle.



# CosTrading Module

Contains the major functional interfaces of a trading service.

## CosTrading Data Types

### CosTrading::Constraint Data Type

```
typedef Istring Constraint;
```

A query constraint expression. The constraint is used to filter offers during a query, and must evaluate to a boolean expression.

The constraint language consists of the following elements:

- comparative functions: `==`, `!=`, `>`, `>=`,
- boolean connectives: `and`, `or`, `not`
- property existence: `exist`
- property names
- numeric, boolean and string constants
- mathematical operators: `+`, `-`, `*`, `/`
- grouping operators: `(`, `)`

The following property value types can be manipulated using the constraint language:

- `boolean`, `short`, `unsigned short`, `long`, `unsigned long`, `float`, `double`, `char`, `lchar`, `string`, `Istring`
- sequences of the above types

Only the `exist` operator can be used on properties of other types.

### Notes

The constraint language keywords are case-sensitive

Literal strings should be enclosed in single quotes

The boolean literals are `TRUE` and `FALSE`

---

## CosTrading::Istring Data Type

```
typedef string Istring;
```

When internationalized strings are widely supported, this definition will be changed.

## CosTrading::LinkName Data Type

```
typedef Istring LinkName;
```

The name of a unidirectional link from one trader to another. The only restriction on the format of a link name is it cannot be an empty string.

## CosTrading::LinkNameSeq Data Type

```
typedef sequence<LinkName> LinkNameSeq;
```

## CosTrading::OfferId Data Type

```
typedef string OfferId;
```

An offer identifier is an opaque string whose format is determined entirely by the trading service from which the offer identifier was obtained, and can only be used with that trading service.

## CosTrading::OfferIdSeq Data Type

```
typedef sequence<OfferId> OfferIdSeq;
```

## CosTrading::OfferSeq Data Type

```
typedef sequence<Offer> OfferSeq;
```

### **CosTrading::PolicyName Data Type**

```
typedef string PolicyName;
```

The name of a policy used to control the trader's behavior. The only restriction on the format of a policy name is it cannot be an empty string.

### **CosTrading::PolicyNameSeq Data Type**

```
typedef sequence<PolicyName> PolicyNameSeq;
```

### **CosTrading::PolicySeq Data Type**

```
typedef sequence<Policy> PolicySeq;
```

### **CosTrading::PolicyValue Data Type**

```
typedef any PolicyValue;
```

### **CosTrading::PropertyName Data Type**

```
typedef Istring PropertyName;
```

Although not explicitly defined in the specification, a property name should start with a letter, may contain digits and underscores, and should not contain spaces.

### **CosTrading::PropertyNameSeq DataType**

```
typedef sequence<PropertyName> PropertyNameSeq;
```

### **CosTrading::PropertySeq Data Type**

```
typedef sequence<Property> PropertySeq;
```

---

## CosTrading::PropertyValue Data Type

```
typedef any PropertyValue;
```

A `CORBA::Any` containing the value of the property. Orbix Trader allows arbitrarily complex user-defined types to be used as property values.

## CosTrading::ServiceTypeName Data Type

```
typedef Istring ServiceTypeName;
```

A service type name can have one of two formats, both representing formats that appear in the Interface Repository.

- **Scoped Name** - A scoped name has the form `::One::Two`. Other supported variations are `Three::Four` and `Five`.
- **Interface Repository Identifier** - An interface repository identifier has the form `IDL:[prefix/][module/]name:X.Y`. For example, `IDL:omg.org/CosTrading/Lookup:1.0` is a valid interface repository identifier, and you can use the same format for your service type names.

---

**Note:** Although a service type name can appear similar to names used in the interface repository, the trading service never uses servicetype names to look up information in the interface repository.

---

## CosTrader::TraderName Data Type

```
typedef LinkNameSeq TraderName;
```

A `TraderName` represents a path from one trader to the desired trader by following a sequence of links. The `starting_trader` importer policy, if specified for a query operation, should contain a value of this type.

## Cos:Trading::TypeRepository Data Type

```
typedef Object TypeRepository;
```



---

`TypeRepository` represents an object reference for a `CosTradingRepos::ServiceTypeRepository` object. You will need to narrow this reference before you can interact with the service type repository.

### CosTrading::FollowOption Enum

```
enum FollowOption
{
    local_only,
    if_no_local,
    always
};
```

Determines the follow behavior for linked traders.

The member values are defined as follows:

<code>local_only</code>	The trader will not follow a link.
<code>if_no_local</code>	The trader will only follow a link if no offers were found locally.
<code>always</code>	The trader will always follow a link.

### CosTrading::Offer Struct

```
struct Offer
{
    Object reference;
    PropertySeq properties;
};
```

The description of a service offer. The data members contains the following data:

<code>reference</code>	The object reference associated with this offer. Depending on the configuration of the server, this reference may be <code>nil</code> .
<code>properties</code>	A sequence of properties associated with this offer.

---

## CosTrading::Policy Struct

```
struct Policy
{
    PolicyName name;
    PolicyValue value;
};
```

## CosTrading::Property Struct

```
struct Property
{
    PropertyName name;
    PropertyValue value;
};
```

A name-value pair associated with a service offer or proxy offer. If the property name matches the name of a property in the offer's service type, then the `TypeCode` of the value must match the property definition in the service type.

---

**Note:** Orbix Trader allows properties to be associated with an offer even if the property name does not match any property in the service type. These properties can also be used in query constraint and preference expressions.

---

# CosTrading Exceptions

## CosTrading::DuplicatePolicyName

```
exception DuplicatePolicyName { PolicyName name};
```

More than one value was supplied for a policy. The policy name that caused the exception is returned.

### **CosTrading::DuplicatePropertyName**

```
exception DuplicatePropertyName {PropertyName name};
```

The property name has already appeared once. The duplicated property name is returned.

### **CosTrading::IllegalConstraint**

```
exception IllegalConstraint{Constraint constr};
```

An error occurred while parsing the constraint expression. The invalid constraint is passed back.

### **CosTrading::IllegalOfferId**

```
exception IllegalOfferId {OfferId id};
```

The offer identifier is empty or malformed. The invalid id is returned.

### **CosTrading::IllegalPropertyName**

```
exception IllegalPropertyName {PropertyName name};
```

The property name is empty or does not conform the format supported by the trader. The property name that caused the exception is returned.

### **CosTrading::IllegalServiceType**

```
exception IllegalServiceType {ServiceTypeName type};
```

A service type name does not conform to the formats supported by the trader. The name that caused the exception is returned.

### **CosTrading::InvalidLookupRef**

```
exception InvalidLookupRef {Lookup target};
```

---

The Lookup object reference cannot be nil.

### **CosTrading::MissingMandatoryProperty**

```
exception MissingMandatoryProperty
{
    ServiceTypeName type;
    PropertyName name;
};
```

No value was supplied for a property defined as mandatory by the service type.

### **CosTrading::NotImplemented**

```
exception NotImplemented {};
```

The requested operation is not supported by this trading service.

### **CosTrading::PropertyTypeMismatch**

```
exception PropertyTypeMismatch
{
    ServiceTypeName type;
    Property prop;
};
```

The property value type conflicts with the property's definition in the service type.

### **CosTrading::ReadOnlyDynamicProperty**

```
exception ReadOnlyDynamicProperty
{
    ServiceTypeName type;
    PropertyName name;
};
```

A property that is defined as read-only by the service type cannot have a dynamic value.

### **CosTrading::UnknownMaxLeft**

```
exception UnknownMaxLeft {};
```

The iterator does not know how many items are left.

### **CosTrading::UnknownOfferId**

```
exception UnknownOfferId {OfferId id};
```

The trader does not contain an offer with the given identifier. The unresolved ID is returned.

### **CosTrading::UnknownServiceType**

```
exception UnknownServiceType {ServiceTypeName type};
```

The service type repository used by the trader does not have the requested service type. The unresolved name is returned.



# CosTrading::Admin Interface

```
// IDL in CosTrading
interface Admin :
  TraderComponents, SupportAttributes,
  ImportAttributes, LinkAttributes
{
  typedef sequence OctetSeq;

  readonly attribute OctetSeq request\_id\_stem;

  unsigned long set\_def\_search\_card (in unsigned long value);
  unsigned long set\_max\_search\_card (in unsigned long value);
  unsigned long set\_def\_match\_card (in unsigned long value);
  unsigned long set\_max\_match\_card (in unsigned long value);
  unsigned long set\_def\_return\_card (in unsigned long value);
  unsigned long set\_max\_return\_card (in unsigned long value);
  unsigned long set\_max\_list (in unsigned long value);

  boolean set\_supports\_modifiable\_properties (in boolean value);
  boolean set\_supports\_dynamic\_properties (in boolean value);
  boolean set\_supports\_proxy\_offers (in boolean value);

  unsigned long set\_def\_hop\_count (in unsigned long value);
  unsigned long set\_max\_hop\_count (in unsigned long value);

  FollowOption set\_def\_follow\_policy (in FollowOption policy);
  FollowOption set\_max\_follow\_policy (in FollowOption policy);
```

---

```

FollowOption set_max_link_follow_policy (
    in FollowOption policy);

TypeRepository set_type_repos (in TypeRepository repository);

OctetSeq set_request_id_stem (in OctetSeq stem);

void list_offers( in unsigned long how_many,
                 out OfferIdSeq ids,
                 out OfferIdIterator id_itr )
raises ( NotImplemented );

void list_proxies( in unsigned long how_many,
                 out OfferIdSeq ids,
                 out OfferIdIterator id_itr )
raises ( NotImplemented );
};

```

Interface `Admin` provides attributes and operations for administrative control of the trading service.

### **Admin::request\_id\_stem Attribute**

readonly attribute OctetSeq request\_id\_stem;

The request identifier “stem” is a sequence of octets that comprise the prefix for a request identifier. The trader will append additional octets to ensure the uniqueness of each request identifier it generates.

### **Admin::list\_offers()**

```

void list_offers(in unsigned long how_many,
                out OfferIdSeq ids,
                out OfferIdIterator id_itr)
raises(NotImplemented);

```

Obtains the identifiers for the service offers in this trader.



---

## Parameters

<code>how_many</code>	Indicates how many identifiers to return in <code>ids</code> .
<code>ids</code>	Contains at most <code>how_many</code> identifiers. If the number of identifiers exceeds <code>how_many</code> , the <code>id_itr</code> parameter will hold a reference to an iterator object through which the remaining identifiers can be obtained.
<code>id_itr</code>	Will hold <code>nil</code> if no identifiers were found or if all of the identifiers were returned in <code>ids</code> . Otherwise, holds a reference to an iterator object through which the remaining identifiers can be obtained.

### **Admin::list\_proxies()**

```
void list_proxies(in unsigned long how_many,  
                 out OfferIdSeq ids,  
                 out OfferIdIterator id_itr)  
raises(NotImplemented);
```

Obtains the identifiers for the proxy offers in this trader.

## Parameters

<code>how_many</code>	Indicates how many identifiers to return in <code>ids</code> .
<code>ids</code>	Contains at most <code>how_many</code> identifiers. If the number of identifiers exceeds <code>how_many</code> , the <code>id_itr</code> parameter will hold a reference to an iterator object through which the remaining identifiers can be obtained.
<code>id_itr</code>	Will hold <code>nil</code> if no identifiers were found or if all of the identifiers were returned in <code>ids</code> . Otherwise, holds a reference to an iterator object through which the remaining identifiers can be obtained.

### **Admin::set\_def\_follow\_policy()**

```
FollowOption set_def_follow_policy(in FollowOption policy);
```

---

Changes the value of the default link follow attribute and returns the previous value.

**Parameters**

policy            The new value

**Admin::set\_def\_hop\_count()**

```
unsigned long set_def_hop_count(in unsigned long value);
```

Changes the value of the default hop count attribute and returns the previous value.

**Parameters**

value            The new value

**Admin::set\_def\_match\_card()**

```
unsigned long set_def_match_card(in unsigned long value);
```

Changes the value of the default match cardinality attribute and returns the previous value.

**Parameters**

value            The new value

**Admin::set\_def\_return\_card()**

```
unsigned long set_def_return_card(in unsigned long value);
```

Changes the value of the default return cardinality attribute and returns the previous value.

**Parameters**

value            The new value

---

### **Admin::set\_def\_search\_card()**

```
unsigned long set_def_search_card(in unsigned long value);
```

Changes the value of the default search cardinality attribute and returns the previous value.

#### **Parameters**

value            The new value

#### **See Also**

`CosTrading::ImportAttributes`

### **Admin::set\_max\_follow\_policy()**

```
FollowOption set_max_follow_policy(in FollowOption policy);
```

Changes the value of the maximum link follow attribute and returns the previous value.

#### **Parameters**

policy           The new value

### **Admin::set\_max\_hop\_count()**

```
unsigned long set_max_hop_count(in unsigned long value);
```

Changes the value of the maximum hop count attribute and returns the previous value.

#### **Parameters**

value            The new value

### **Admin::set\_max\_link\_follow\_policy()**

```
FollowOption set_max_link_follow_policy(in FollowOption policy);
```

Changes the value of the maximum link follow policy and returns the previous value.

---

## Parameters

policy            The new value

### **Admin::set\_max\_list()**

```
unsigned long set_max_list(in unsigned long value);
```

Changes the value of the maximum list attributes and returns the previous value.

## Parameters

value            The new value

### **Admin::set\_max\_match\_card()**

```
unsigned long set_max_match_card(in unsigned long value);
```

Changes the value of the maximum match cardinality attribute and returns the previous value.

## Parameters

value            The new value

### **Admin::set\_max\_return\_card()**

```
unsigned long set_max_return_card(in unsigned long value);
```

Changes the value of the maximum return cardinality attribute and returns the previous value.

## Parameters

value            The new value

---

### **Admin::set\_max\_search\_card()**

`unsigned long set_max_search_card(in unsigned long value);`

Changes the value of the maximum search cardinality attribute and returns the previous value.

#### **Parameters**

value                    The new value

### **Admin::set\_request\_id\_stem()**

`OctetSeq set_request_id_stem(in OctetSeq stem);`

Changes the value of the request identifier stem and returns the previous value.

#### **Parameters**

stem                    The new value

### **Admin::set\_supports\_dynamic\_properties()**

`boolean set_supports_dynamic_properties(in boolean value);`

Establishes whether the trader considers offers with dynamic properties during a query and returns the previous setting.

#### **Parameters**

value                    The new value

### **Admin::set\_supports\_modifiable\_properties()**

`boolean set_supports_modifiable_properties(in boolean value);`

Establishes whether the trader supports property modification and returns the previous setting.

---

## Parameters

- value
- `TRUE` activates property modification support.
  - `FALSE` deactivates property modification support.

### **Admin::set\_supports\_proxy\_offers()**

```
boolean set_supports_proxy_offers(in boolean value);
```

Establishes whether the trader supports proxy offers and returns the previous setting.

## Parameters

- value
- `TRUE` turns on proxy support.
  - `FALSE` turns off proxy support.

### **Admin::set\_type\_repos()**

```
TypeRepository set_type_repos(in TypeRepository repository);
```

Establishes the service type repository to be used by the trader and returns a reference to the previous type repository.

## Parameters

- repository
- A reference to a type repository.

# CosTrading::ImportAttributes Interface

The read-only attributes of this interface provide the default and maximum values for policies that govern query operations.

---

**Note:** Performing a query is also known as *importing service offers*, therefore these attributes are called *import attributes*.

---

## ImportAttributes::def\_follow\_policy Attribute

readonly attribute [FollowOption](#) def\_follow\_policy;

The default value for the `follow_policy` policy if it is not supplied.

## ImportAttributes::def\_hop\_count Attribute

readonly attribute unsigned long def\_hop\_count;

The default value for the `hop_count` policy if it is not supplied.

## ImportAttributes::def\_match\_card Attribute

readonly attribute unsigned long def\_match\_card;

The default value for the `match_card` policy if it is not supplied.

## ImportAttributes::def\_return\_card Attribute

readonly attribute unsigned long def\_return\_card;

---

The default value for the `return_card` policy if it is not supplied.

### **ImportAttributes::def\_search\_card Attribute**

readonly attribute unsigned long def\_search\_card;

The default value for the `search_card` policy if it is not supplied.

### **ImportAttributes::max\_follow\_policy Attribute**

readonly attribute [FollowOption](#) max\_follow\_policy;

The maximum value for the `follow_policy` policy, which may override the value supplied by an importer.

### **ImportAttributes::max\_hop\_count Attribute**

readonly attribute unsigned long max\_hop\_count;

The maximum value for the `hop_count` policy, which may override the value supplied by an importer.

### **ImportAttributes::max\_list Attribute**

readonly attribute unsigned long max\_list;

The maximum size of any list returned by the trader. This may override the value supplied by a client to operations such as `query` and `next_n`.

### **ImportAttributes::max\_match\_card Attribute**

readonly attribute unsigned long max\_match\_card;

The maximum value for the `match_card` policy, which may override the value supplied by an importer.



---

### **ImportAttributes::max\_return\_card Attribute**

`readonly attribute unsigned long max_return_card;`

The maximum value for the `return_card` policy, which may override the value supplied by an importer.

### **ImportAttributes::max\_search\_card Attribute**

`readonly attribute unsigned long max_search_card;`

The maximum value for the `search_card` policy, which may override the value supplied by an importer.



# CosTrading::Link Interface

```
interface Link :
    TraderComponents, SupportAttributes, LinkAttributes
{
    struct LinkInfo
    {
        Lookup target;
        Register target_reg;
        FollowOption def_pass_on_follow_rule;
        FollowOption limiting_follow_rule;
    };

    exception IllegalLinkName { LinkName name; };
    exception UnknownLinkName { LinkName name; };
    exception DuplicateLinkName { LinkName name; };

    exception DefaultFollowTooPermissive {
        FollowOption default_follow_rule;
        FollowOption limiting_follow_rule; };
    exception LimitingFollowTooPermissive {
        FollowOption limiting_follow_rule;
        FollowOption max_link_follow_policy; };

    void add\_link( in LinkName name, in Lookup target,
                 in FollowOption default_follow_rule,
                 in FollowOption limiting_follow_rule )
    raises ( IllegalLinkName, DuplicateLinkName, InvalidLookupRef,
            DefaultFollowTooPermissive,
            LimitingFollowTooPermissive );

    void remove\_link( in LinkName name )
    raises ( IllegalLinkName, UnknownLinkName );

    LinkInfo describe\_link( in LinkName name )
    raises ( IllegalLinkName, UnknownLinkName );

    LinkNameSeq list\_links();
}
```

---

```

void modify\_link( in LinkName name,
                  in FollowOption default_follow_rule,
                  in FollowOption limiting_follow_rule )
raises ( IllegalLinkName, UnknownLinkName,
        DefaultFollowTooPermissive,
        LimitingFollowTooPermissive );
};

```

Provides structures, exceptions, and operations for managing links between traders.

## Link::LinkInfo Data Structure

```

struct LinkInfo
{
    Lookup target;
    Register target_reg;
    FollowOption def_pass_on_follow_rule;
    FollowOption limiting_follow_rule;
};

```

A complete description of a link. The members hold the following information:

target	Lookup interface if link target
target_reg	Register interface of link
def_pass_on_follow_rule	Default link behavior for the link if no link-follow policy is specified by an importer during a query
limiting_follow_rule	Most permissive link-follow behavior that the link is willing to tolerate

## CosTrading::Link Exceptions

### Link::DefaultFollowTooPermissive Exception

```

exception DefaultFollowTooPermissive
{

```

```
    FollowOption def_pass_on_follow_rule;  
    FollowOption limiting_follow_rule;  
};
```

Raised when the value for `def_pass_on_follow_rule` exceeds the value for `limiting_follow_rule`. Both values are passed back to the caller.

### Link::DuplicateLinkName Exception

```
exception DuplicateLinkName {LinkName name};
```

Raised when a link already exists with the given name. The duplicated link name is passed back to the caller.

### Link::IllegalLinkName Exception

```
exception IllegalLinkName {LinkName name};
```

Raised when the link name is empty or does not conform the format supported by the trader. The invalid link name is passed back to the caller.

### Link::LimitingFollowTooPermissive Exception

```
exception LimitingFollowTooPermissive  
{  
    FollowOption limiting_follow_rule;  
    FollowOption max_link_follow_policy;  
};
```

The value for `limiting_follow_rule` exceeds the trader's `max_link_follow_policy` attribute.

### Link::UnknownLinkName Exception

```
exception UnknownLinkName {LinkName name};
```

Raised when trader does not have a link with the given name. The invalid name is returned.

---

## Link::add\_link()

```
void add_link(in LinkName name, in Lookup target,  
             in FollowOption def_pass_on_follow_rule,  
             in FollowOption limiting_follow_rule)  
raises(IllegalLinkName,  
       DuplicateLinkName,  
       InvalidLookupRef,  
       DefaultFollowTooPermissive,  
       LimitingFollowTooPermissive);
```

Adds a new, unidirectional link from this trader to another trader.

### Parameters

name	Specifies the name of the new link.
target	Holds a reference to the <code>Lookup</code> interface of the target trader
def_pass_on_follow_rule	Specifies the default link behavior for the link if not link-follow policy is specified by an importer during a query.
limiting_follow_rule	Specifies the most permissive link-follow behavior the the link is willing to follow.

### Exceptions

<code>IllegalLinkName</code>	Link name is empty or has an invalid format.
<code>DuplicateLinkName</code>	Another link exists with the same name.
<code>InvalidLookupRef</code>	Target object reference is nil.
<code>DefaultFollowTooPermissive</code>	The value for <code>def_pass_on_follow_rule</code> exceeds the value for <code>limiting_follow_rule</code> .
<code>LimitingFollowTooPermissive</code>	The value for <code>limiting_follow_rule</code> exceeds the trader's <a href="#">max_link_follow_policy</a> .

### Link::describe\_link()

```
LinkInfo describe_link(in LinkName name)
raises(IllegalLinkName, UnknownLinkName);
```

Obtains a description of a link and returns it in a [LinkInfo](#) object.

#### Parameters

name                    Name of the link of interest

#### Exceptions

[IllegalLinkName](#) The link name is empty or has an invalid format.

[UnknownLinkName](#) No link with the specified name exists.

### Link::list\_links()

```
LinkNameSeq list_links();
```

Returns the names of all trading links within the trader.

### Link::modify\_link()

```
void modify_link(in LinkName name,
                 in FollowOption def_pass_on_follow_rule,
                 in FollowOption limiting_follow_rule)
raises(IllegalLinkName,
      UnknownLinkName,
      DefaultFollowTooPermissive,
      LimitingFollowTooPermissive);
```

Modifies the follow behavior of an existing link.

#### Parameters

name                    Specifies the name of the link to be modified.

---

<code>def_pass_on_follow_rule</code>	Specifies the default link behavior for the link if no link-follow policy is specified by an importer during a query.
<code>limiting_follow_rule</code>	Describes the most permissive link-follow behavior that the link is willing to follow.

## Exceptions

<code>IllegalLinkName</code>	Link name is empty or has an invalid format.
<code>UnknownLinkName</code>	The specified link name does not exist.
<code>DefaultFollowTooPermissive</code>	The value for <code>def_pass_on_follow_rule</code> exceeds the value for <code>limiting_follow_rule</code> .
<code>LimitingFollowTooPermissive</code>	The value for <code>limiting_follow_rule</code> exceeds the trader's <a href="#">max_link_follow_policy</a> .

## Link::remove\_link()

```
void remove_link(in LinkName name)
    raises(IllegalLinkName, UnknownLinkName);
```

Removes an existing link.

## Parameters

<code>name</code>	Name of the link to be removed
-------------------	--------------------------------

## Exceptions

<code>IllegalLinkName</code>	The link name is empty or has an invalid format.
<code>UnknownLinkName</code>	No link exists with the specified name.



# CosTrading::LinkAttributes Interface

## LinkAttributes::max\_link\_follow\_policy Attribute

readonly attribute [FollowOption](#) max\_link\_follow\_policy;

Determines the most permissive behavior that will be allowed for any link.



# CosTrading::Lookup Interface

```
interface Lookup :
    TraderComponents, SupportAttributes, ImportAttributes
{
    typedef Istring Preference;

    enum HowManyProps
    {
        none,
        some,
        all
    };

    union SpecifiedProps switch (HowManyProps)
    {
        case some: PropertyNameSeq prop_names;
    };

    exception IllegalPreference {Preference pref};
    exception IllegalPolicyName {PolicyName name};
    exception PolicyTypeMismatch {Policy the_policy};
    exception InvalidPolicyValue {Policy the_policy};

    void query(in ServiceTypeName type,
              in Constraint constr,
              in Preference pref,
              in PolicySeq policies,
              in SpecifiedProps desired_props,
              in unsigned long how_many,
              out OfferSeq offers,
              out OfferIterator offer_itr,
              out PropertyNameSeq limits_applied)
    raises (IllegalServiceType, UnknownServiceType,
           IllegalConstraint, IllegalPreference,
           IllegalPolicyName, PolicyTypeMismatch,
           InvalidPolicyValue, IllegalPropertyName,
           DuplicatePropertyName, DuplicatePolicyName);
};
```

---

Provides a single operation, `query`, for use by importers.

## Lookup::Preference DataType

```
typedef Istring Preference;
```

A query preference expression. The preference is used to order the offers found by a query. The valid forms of a preference expression are:

**min *numeric-expression*** orders the offers in ascending order based on the numeric expression. Offers for which the expression cannot be evaluated (for example, if the offer does not contain a property that is used in the expression) are placed at the end of the sequence.

**max *numeric-expression*** orders the offers in descending order based on the numeric expression. Offers for which the expression cannot be evaluated (for example, if the offer does not contain a property that is used in the expression) are placed at the end of the sequence.

**with *boolean-expression*** orders the offers such that those for which the boolean expression are `TRUE` are included before any of those for which the expression is false, which are placed before any of those that cannot be evaluated.

**random** orders the offers in random order.

**first** orders the offers as they are encountered by the server.

If an empty preference expression is supplied, it is equivalent to a preference of `first`.

## Lookup::HowManyProps Enum

```
enum HowManyProps  
{  
    none,  
    some,  
    all
```

---

```
};
```

The choices for indicating how many properties are returned with each offer. The members are defined as follows:

<code>none</code>	No properties should be returned.
<code>some</code>	Some properties should be returned.
<code>all</code>	All properties should be returned.

### Lookup::SpecifiedProps Union

```
union SpecifiedProps switch(HowManyProps)
{
case some: PropertyNameSeq prop_names;
};
```

Determines which properties are to be returned for each matching offer found by the [query](#) operation. The union's discriminator can meaningfully be set to the other enumerated values `none` and `all`. If set to `none`, you are indicating that no properties should be returned. If set to `all`, then all properties will be returned. Set the value for `some` with a sequence of property names indicating which properties should be returned

### Lookup::IllegalPolicyName Exception

```
exception IllegalPolicyName (PolicyName name);
```

The policy name is empty or does not conform the format supported by the trader. The invalid name is returned.

### Lookup::IllegalPreference Exception

```
exception IllegalPreference (Preference pref);
```

An error occurred while parsing the preference expression. The invalid preference is returned.

---

## Lookup::InvalidPolicyValue Exception

```
exception InvalidPolicyValue {Policy the_policy};
```

The policy has an invalid value.

## Lookup::PolicyTypeMismatch Exception

```
exception PolicyTypeMismatch {Policy the_policy};
```

The policy value type specified does not match the type expected by the trader. The type expected by the trader is returned.

## Lookup::query()

```
void query(in ServiceTypeName type,  
           in Constraint constr,  
           in Preference pref,  
           in PolicySeq policies,  
           in SpecifiedProps desired_props,  
           in unsigned long how_many,  
           out OfferSeq offers,  
           out OfferIterator offer_itr,  
           out PolicyNameSeq limits_applied)  
raises(IllegalServiceType,  
       UnknownServiceType,  
       IllegalConstraint,  
       IllegalPreference,  
       IllegalPolicyName,  
       PolicyTypeMismatch,  
       InvalidPolicyValue,  
       IllegalPropertyName,  
       DuplicatePropertyName,  
       DuplicatePolicyName);
```

Allows an *importer* to obtain references to objects that provide services meeting its requirements.

The importer can control the behavior of the search by supplying values for certain policies. The trader may override some or all of the values supplied by the importer. The following policies are known by the trader:

---

**exact\_type\_match (boolean)** if `TRUE`, only offers of exactly the service type specified by the importer are considered; if `FALSE`, offers of any service type that conforms to the importer's service type are considered

**hop\_count (unsigned long)** indicates maximum number of hops across federation links that should be tolerated in the resolution of this query

**link\_follow\_rule (FollowOption)** indicates how the client wishes links to be followed in the resolution of this query

**match\_card (unsigned long)** indicates the maximum number of matching offers to which the preference specification should be applied

**return\_card (unsigned long)** indicates the maximum number of matching offers to return as a result of this query

**search\_card (unsigned long)** indicates the maximum number of offers to be considered when looking for type conformance and constraint expression match

**starting\_trader (TraderName)** specifies the remote trader at which the query starts

**use\_dynamic\_properties (boolean)** specifies whether to consider offers with dynamic properties

**use\_modifiable\_properties (boolean)** specifies whether to consider offers with modifiable properties

**use\_proxy\_offers (boolean)** specifies whether to consider proxy offers

---

## Parameters

<code>type</code>	Specifies the service type that interests the importer. The service type limits the scope of the search to only those offers exported for this type, and optionally any subtype of this type.
<code>constr</code>	Limits the search to only those offers for which this expression is <code>TRUE</code> . The simplest constraint expression is "TRUE", which matches any offer.
<code>pref</code>	Specifies how the matched offers are to be ordered.
<code>policies</code>	Specifies the policies that govern the behavior of the query.
<code>desired_props</code>	Determines the properties that are to be included with each offer returned by the query. This parameter does not affect whether or not a service offer is returned. To exclude an offer that does not contain a desired property, include "exist <i>property-name</i> " in the constraint.
<code>how_many</code>	Indicates how many offers are to be returned in the <code>offers</code> parameter.
<code>offers</code>	Holds at most <code>how_many</code> offers. If the number of matching offers exceeds <code>how_many</code> , the <code>offer_itr</code> parameter will hold a reference to an iterator object through which the remaining offers can be obtained.
<code>offer_itr</code>	Will hold <code>nil</code> if no matching offers were found or if all of the matching offers were returned in <code>offers</code> ; otherwise, holds a reference to an iterator. The object's <code>destroy</code> operation should be invoked when the object is no longer needed.
<code>limits_applied</code>	Holds the names of any policies that were overridden by the trader's maximum allowable settings.

## Exceptions

<a href="#"><u>IllegalServiceType</u></a>	Service type name is empty or has an invalid format
<a href="#"><u>UnknownServiceType</u></a>	Service type was not found in service type repository
<a href="#"><u>IllegalConstraint</u></a>	An error occurred while parsing the constraint expression



---

<a href="#"><u>IllegalPreference</u></a>	An error occurred while parsing the preference expression
<a href="#"><u>IllegalPolicyName</u></a>	A policy name is empty or has an invalid format
<a href="#"><u>PolicyTypeMismatch</u></a>	A policy value type did not match the type expected by the trader
<a href="#"><u>InvalidPolicyValue</u></a>	A policy has an invalid value
<a href="#"><u>IllegalPropertyName</u></a>	A property name is empty or has an invalid format
<a href="#"><u>DuplicatePropertyName</u></a>	A property name appeared more than once in the list of desired properties
<a href="#"><u>DuplicatePolicyName</u></a>	A policy name appeared more than once in the list of policies



# CosTrading::OfferIdIterator Interface

```
interface OfferIdIterator
{
    unsigned long max\_left\(\)
    raises (UnknownMaxLeft);

    boolean next\_n(in unsigned long n, out OfferIdSeg ids);

    void destroy\(\);
};
```

Specifies methods to iterate through a list of offer identifiers.

## **OfferIdIterator::destroy()**

```
void destroy();
```

Destroys the iterator object.

## **OfferIdIterator::max\_left()**

```
unsigned long max_left()
raises(UnknownMaxLeft);
```

Returns the number of offer identifiers remaining in the iterator.

## **Exceptions**

[UnknownMaxLeft](#) Cannot determine the number of remaining offer identifiers

## **OfferIdIterator::next\_n()**

```
boolean next_n(in unsigned long n,
               out OfferIdSeg ids);
```

---

Returns `TRUE` if `ids` contains more offer identifiers, and returns `FALSE` if `ids` is `nil`.

**Parameters**

<code>n</code>	Number of offer identifiers to return
<code>ids</code>	List of offer identifiers containing at most <code>n</code> elements

# CosTrading::OfferIterator Interface

```
interface OfferIterator
{
    unsigned long max\_left\(\)
    raises (UnknownMaxLeft);

    boolean next\_n( in unsigned long n, out OfferSeq offers );

    void destroy();
};
```

Specifies methods to iterate through a list of offers.

## OfferIterator::destroy()

```
void destroy();
```

Destroys the iterator object.

## OfferIterator::max\_left()

```
unsigned long max_left()
raises(UnknownMaxLeft);
```

Returns the number of offers remaining in the iterator.

## Exceptions

[UnknownMaxLeft](#) cannot determine the number of remaining offers

## OfferIterator::next\_n()

```
boolean next_n(in unsigned long n,
               out OfferSeq offers);
```

---

Returns `TRUE` if `offers` contains more offer identifiers, and returns `FALSE` if `offers` is `nil`.

**Parameters**

<code>n</code>	Number of offers to return
<code>ids</code>	List of offers containing at most <code>n</code> elements

# CosTrading::Proxy Interface

```
interface Proxy :
    TraderComponents,
    SupportAttributes
{
    typedef Istring ConstraintRecipe;

    struct ProxyInfo
    {
        ServiceTypeName type;
        Lookup target;
        PropertySeq properties;
        boolean if_match_all;
        ConstraintRecipe recipe;
        PolicySeq policies_to_pass_on;
    };

    exception IllegalRecipe {ConstraintRecipe recipe};
    exception NotProxyOfferId {OfferId id};

    OfferId export\_proxy(in Lookup target, in ServiceTypeName type,
                        in PropertySeq properties,
                        in boolean if_match_all,
                        in ConstraintRecipe recipe,
                        in PolicySeq policies_to_pass_on)
    raises (IllegalServiceType, UnknownServiceType,
           InvalidLookupRef, IllegalPropertyName,
           PropertyTypeMismatch, ReadOnlyDynamicProperty,
           MissingMandatoryProperty, IllegalRecipe,
           DuplicatePropertyName, DuplicatePolicyName);

    void withdraw\_proxy( in OfferId id )
    raises (IllegalOfferId, UnknownOfferId, NotProxyOfferId);

    ProxyInfo describe\_proxy( in OfferId id )
    raises (IllegalOfferId, UnknownOfferId, NotProxyOfferId);
};
```

---

Provides datatypes, exceptions and methods for managing proxy offers.

## Proxy::ConstraintRecipe Data Type

```
typedef Istring ConstraintRecipe;
```

A constraint recipe specifies how the trader should rewrite a constraint before invoking the `query` operation of the proxy offer's [Lookup](#) interface. Using a constraint recipe, the exporter can have the trader rewrite a constraint into a completely different constraint language (one that is understood by the proxy offer's [Lookup](#) target).

The constraint recipe can include the value of properties using the expression "`$(property-name)`". The recipe can also include the entire text of the original constraint using the special syntax "\$\*".

For example, assume the property `name` has the value "Joe", and the property `age` has the value 33. The constraint recipe "Name == \$(name) and Age" would be rewritten as "Name == 'Joe' and Age".

## Proxy::ProxyInfo Data Structure

```
struct ProxyInfo
{
    ServiceTypeName type;
    Lookup target;
    PropertySeq properties;
    boolean if_match_all;
    ConstraintRecipe recipe;
    PolicySeq policies_to_pass_on;
};
```

A complete description of a proxy offer which contains the following members:

<code>type</code>	The service type for which tis offer was exported.
<code>target</code>	The target <a href="#">Lookup</a> object.
<code>properties</code>	A sequence of properties associated with this offer.



---

<code>if_match_all</code>	If <code>TRUE</code> , type conformance is all that is necessary for this offer to match. If <code>FALSE</code> , the offer must also match the constraint expression.
<code>recipe</code>	The recipe for rewriting the constraint
<code>policies_to_pass_on</code>	Policies to be appended to the importer's policies and passed along to the target.

### Proxy::IllegalRecipe Exception

```
exception IllegalRecipe(ConstraintRecipe recipe);
```

An error occurred while parsing the recipe.

### Proxy::NotProxyOfferId Exception

```
exception NotProxyOfferId(OfferId id);
```

The offer identifier does not refer to a proxy offer.

### Proxy::describe\_proxy()

```
ProxyInfo describe_proxy(in OfferId id)
raises(IllegalOfferId,
       UnknownOfferId,
       NotProxyOfferId);
```

Obtains the description of a proxy offer.

#### Parameters

<code>id</code>	Identifier of the proxy offer of interest
-----------------	---

#### Exceptions

`IllegalOfferId` Offer Identifier is empty or has an invalid format.

`UnknownOfferId` No offer was found with the given identifier

`NotProxyOfferId` Offer identifier does not refer to a proxy offer

---

## Proxy::export\_proxy()

```
OfferId export_proxy(in Lookup target,  
                    in ServiceTypeName type,  
                    in PropertySeq properties,  
                    in boolean if_match_all,  
                    in ConstraintRecipe recipe,  
                    in PolicySeq policies_to_pass_on)  
raises(IllegalServiceType,  
       UnknownServiceType,  
       InvalidLookupRef,  
       IllegalPropertyName,  
       PropertyTypeMismatch,  
       ReadonlyDynamicProperty,  
       MissingMandatoryProperty,  
       IllegalRecipe,  
       DuplicatePropertyName,  
       DuplicatePolicyName);
```

Creates a new proxy offer.

### Parameters

target	The target <a href="#">Lookup</a> interface
type	The service type for which this offer was exported
properties	A sequence of properties associated with this offer.
if_match_all	If <code>TRUE</code> , type conformance is all that is necessary for this offer to match. If <code>FALSE</code> , the offer must also match the constraint expression.
recipe	The recipe for rewriting the constraint.
policies_to_pass_on	Policies to be appended to the importer's policies and passed along to the target.

### Exceptions

<a href="#">IllegalServiceType</a>	Service type name is empty or has invalid format.
<a href="#">UnknownServiceType</a>	Service type was not found in the service type repository.
<a href="#">InvalidLookupRef</a>	Target object reference is <code>nil</code> .

---

<code>IllegalPropertyName</code>	Property name is empty or has an invalid format.
<code>PropertyTypeMismatch</code>	Property value type does not match the property definition of the service type.
<code>ReadOnlyDynamicProperty</code>	Read-only properties cannot have dynamic values.
<code>MissingMandatoryProperty</code>	No value was given for a mandatory property.
<code>IllegalRecipe</code>	An error occurred while parsing the constraint recipe.
<code>DuplicatePropertyname</code>	A property name appeared more than once in the list of properties.
<code>DuplicatePolicyName</code>	A policy name appeared more than once in the list of policies to pass on.

### **Proxy::withdraw\_proxy()**

```
void withdraw_proxy(in OfferId id)
    raises(IllegalOfferId,
          UnknownOfferId,
          NotProxyOfferId);
```

Removes a proxy offer.

#### **Parameters**

`id` Identifier of the proxy offer to be withdrawn

#### **Exceptions**

`IllegalOfferId` Offer identifier is empty or has an invalid format

`UnknownOfferId` No offer was found with the given identifier.

`NotProxyOfferId` Offer identifier does not refer to a proxy offer



# CosTrading::Register Interface

```
interface Register
inherits from CosTrading::TraderComponents, CosTrading::SupportAttributes
```

Provides operations for managing service offers.

## Register::OfferInfo Structure

```
struct OfferInfo
{
    Object reference;
    ServiceTypeName type;
    PropertySeq properties;
};
```

A complete description of a service offer.

reference	The object reference associated with this offer. Depending on the configuration of the server, this reference may be <code>nil</code> .
type	The service type for which this offer was exported
properties	A sequence of properties associated with this offer.

## Register::IllegalTraderName Exception

```
exception IllegalTraderName
{
    TraderName name;
};
```

The trader name was empty, or a component of the name was not a valid link name.

---

## Register::InterfaceTypeMismatch Exception

```
exception InterfaceTypeMismatch
{
    ServiceTypeName type;
    Object reference;
};
```

If the trader is configured to use the interface repository, then it will attempt to confirm that the interface of the object reference conforms to the interface of the service type. If the trader is able to determine that there is a mismatch, this exception is thrown.

## Register::InvalidObjectRef Exception

```
exception InvalidObjectRef
{
    Object ref;
};
```

The object reference is `nil`, and the trader is configured to reject offers with `nil` references.

## Register::MandatoryProperty Exception

```
exception MandatoryProperty
{
    ServiceTypeName type;
    PropertyName name;
};
```

A mandatory property cannot be removed.

## Register::NoMatchingOffers Exception

```
exception NoMatchingOffers
{
    Constraint constr;
};
```

---

No matching offers were found matching the constraint expression.

### **Register::ProxyOfferId Exception**

```
exception ProxyOfferId
{
    OfferId id;
};
```

The offer identifier actually refers to a proxy offer.

### **Register::ReadOnlyProperty Exception**

```
exception ReadOnlyProperty
{
    ServiceTypeName type;
    PropertyName name;
};
```

A read-only property cannot be modified.

### **Register::RegisterNotSupported Exception**

```
exception RegisterNotSupported
{
    TraderName name;
};
```

The resolve operation is not supported by this trader.

### **Register::UnknownPropertyName Exception**

```
exception UnknownPropertyName
{
    PropertyName name;
};
```

A property was identified for removal that does not exist in the offer.

---

## Register::UnknownTraderName Exception

```
exception UnknownTraderName
{
    TraderName name;
};
```

The trader name could not be correctly resolved to a trader.

## Register::describe()

```
OfferInfo describe(in OfferId id)
    raises(IllegalOfferId,
          UnknownOfferId,
          ProxyOfferId);
```

Obtains the description of a service offer and returns it in an [OfferInfo](#) structure.

### Parameters

id Identifier of the offer of interest

### Exceptions

[IllegalOfferId](#) Offer identifier is empty or has an invalid format

[UnknownOfferId](#) No offer was found with the given identifier

[ProxyOfferId](#) Offer identifier refers to a proxy offer. Proxy offers must be described using the [Proxy](#) interface.

## Register::export()

```
OfferId export(in Object reference,
               in ServiceTypeName type,
               in PropertySeq properties)
    raises(InvalidObjectRef,
          IllegalServiceType,
          UnknownServiceType,
          InterfaceTypeMismatch,
          IllegalPropertyName,
```



---

```
PropertyTypeMismatch,  
ReadOnlyDynamicProperty,  
MissingMandatoryProperty,  
DuplicatePropertyName);
```

Creates a new service offer and returns an identifier object for the new service. A client wishing to advertise a new offer is called an *exporter*.

### Parameters

<code>reference</code>	Reference to an object that enables a client to interact with a remote server.
<code>type</code>	Identifies the service type for which this offer is advertised.
<code>properties</code>	List of named values that describe the service being offered.

### Exceptions

<code>InvalidObjectRef</code>	Object reference is <code>nil</code> and the trader has been configured to reject <code>nil</code> references
<code>IllegalServiceType</code>	Service type name is empty or has an invalid format
<code>UnknownServiceType</code>	Service type was not found in service type repository
<code>InterfaceTypeMismatch</code>	Trader was able to determine that the interface of the object reference does not conform to the the interface of the service type
<code>IllegalPropertyName</code>	Property name is empty or has an invalid format
<code>PropertyTypeMismatch</code>	Property value type does not match the property definition of the service type
<code>ReadOnlyDynamicProperty</code>	Read-only properties cannot have dynamic values
<code>MissingMandatoryProperty</code>	No value was supplied for a mandatory property
<code>DuplicatePropertyName</code>	Property name appeared more than once in list of properties

---

## Register::modify()

```
void modify(in OfferId id,
            in PropertyNameSeq del_list,
            in PropertySeq modify_list)
    raises(NotImplemented,
           IllegalOfferId,
           UnknownOfferId,
           ProxyOfferId,
           IllegalPropertyName,
           UnknownPropertyName,
           PropertyTypeMismatch,
           ReadonlyDynamicProperty,
           MandatoryProperty,
           ReadonlyProperty,
           DuplicatePropertyName);
```

Modifies an existing service offer to add new properties, and change or delete existing properties.

### Parameters

id	Identifier of the offer to be modified
del_list	Names of properties to be removed
modify_list	Properties to be added or modified

### Exceptions

NotImplemented	Trader does not support modification of properties
IllegalOfferId	Offer identifier is empty or has an invalid format
UnknownOfferId	No offer was found with the given identifier
ProxyOfferId	Offer identifier refers to a proxy offer. Proxy offers must be described using the <code>Proxy</code> interface.
IllegalPropertyName	Property name is empty or has an invalid format
UnknownPropertyName	Property to be removed does not exist in offer
PropertyTypeMismatch	Property value type does not match the property definition of the service type
ReadonlyDynamicProperty	Read-only properties cannot have dynamic values

---

MandatoryProperty	Mandatory properties cannot be removed
ReadOnlyProperty	Read-only properties cannot be modified
DuplicatePropertyName	Property name appeared more than once in list of properties

### Register::resolve()

```
Register resolve(in TraderName name)
    raises(IllegalTraderName,
           UnknownTraderName,
           RegisterNotSupported);
```

Resolves a context-relative name for another trader and returns a Register object for the resolved trader.

#### Parameters

name Identifies the trader to be resolved

#### Exceptions

IllegalTraderName Trader name was empty, or a component of the name was not a valid link name

UnknownTraderName Trader name could not be correctly resolved to a trader

RegisterNotSupportedTrader does not support this operation

### Register::withdraw()

```
void withdraw(in OfferId id)
    raises(IllegalOfferId,
           UnknownOfferId,
           ProxyOfferId);
```

Removes a service offer.

#### Parameters

id Identifier of the offer to be withdrawn

---

## Exceptions

`IllegalOfferId` Offer identifier is empty or has an invalid format  
`UnknownOfferId` No offer was found with the given identifier  
`ProxyOfferId` Offer identifier refers to a proxy offer. Proxy offers must be removed using the `Proxy` interface.

## Register::withdraw\_using\_constraint()

```
void withdraw_using_constraint(in ServiceTypeName type,  
                              in Constraint constr)  
    raises(IllegalServiceType,  
          UnknownServiceType,  
          IllegalConstraint,  
          NoMatchingOffers);
```

Withdraws all offers for a particular service type that match a constraint expression. Only offers that exactly match the given service type are considered. Proxy offers are not considered, and links are not followed.

## Parameters

<code>type</code>	Identifies the service type for which offers are to be removed.
<code>constr</code>	Limits the search to only those offers for which this expression is true. The simplest constraint expression is <code>TRUE</code> , which matches any offer and is an efficient way to withdraw all offers for a service type.

## Exceptions

`IllegalServiceType` Service type name is empty or has an invalid format  
`UnknownServiceType` Service type was not found in service type repository  
`IllegalConstraint` An error occurred while parsing the constraint expression  
`NoMatchingOffers` No matching offers were found

# CosTrading::SupportAttributes Interface

```
interface SupportAttributes
```

The read-only attributes in this interface determine what additional functionality a trader supports, and also provide access to the service type repository used by the trader.

## **SupportAttributes::supports\_dynamic\_properties Attribute**

```
readonly attribute boolean supports_dynamic_properties;
```

If `FALSE`, offers with dynamic properties will not be considered during a query.

## **SupportAttributes::supports\_modifiable\_properties Attribute**

```
readonly attribute boolean supports_modifiable_properties;
```

If `FALSE`, the `modify` operation of the `Register` interface will raise `NotImplemented`.

## **SupportAttributes::supports\_proxy\_offers Attribute**

```
readonly attribute boolean supports_proxy_offers;
```

If `FALSE`, the `proxy_if` attribute of the `TraderComponents` interface will return `nil`, and proxy offers will not be considered during a query.

## **SupportAttributes::type\_repos Attribute**

```
readonly attribute TypeRepository type_repos;
```

---

Returns the object reference of the service type repository used by the trader.

# CosTrading::TraderComponents Interface

```
interface TraderComponents
```

Each of the five major interfaces of the `CosTrading` module inherit from this interface. By doing so, any of the *trader components* can be obtained using a reference to any of the other components.

A `nil` value will be returned by an attribute if the trader does not support that interface.

## TraderComponents::admin\_if Attribute

```
readonly attribute Admin admin_if;
```

## TraderComponents::link\_if Attribute

```
readonly attribute Link link_if;
```

## TraderComponents::lookup\_if Attribute

```
readonly attribute Lookup lookup_if;
```

## TraderComponents::proxy\_if Attribute

```
readonly attribute Proxy proxy_if;
```

---

## TraderComponents::register\_if Attribute

readonly attribute [Register](#) register\_if;



# CosTrading::Dynamic Module

Defines interfaces and types necessary to support dynamic properties. Dynamic properties allow an exporter to delegate a property's value to a third party. For example, rather than exporting an offer with a value of 54 for the property `weight`, you can provide a reference to an object that will dynamically compute the value for `weight`.

Naturally, there are performance issues when using dynamic properties, and therefore an importer may elect to exclude any offers containing dynamic properties.

To export an offer (or a proxy offer) with a dynamic property, you need to do the following:

- Define an object that implements the `DynamicPropEval` interface.
- Create an instance of the `DynamicProp` struct and insert that into the property's `CORBA::Any` value.
- Ensure that the lifetime of the `DynamicPropEval` object is such that it will be available whenever dynamic property evaluation is necessary.

## CosTradingDynamic::DynamicProp Struct

```
struct DynamicProp
{
    DynamicPropEval eval_if;
    TypeCode returned_type;
    any extra_info;
};
```

Describes a dynamic property. This struct is inserted into a property's `CORBA::Any` value and provides all of the information necessary for the trader to accomplish dynamic property evaluation.

`eval_if`                      Object reference for evaluation interface

---

<code>returned_type</code>	Value type expected for the property. The value of <code>returned_type</code> must match the value type of the property as defined by the service type.
<code>extra_info</code>	Additional information used for property evaluation. Orbix Trader supports primitive and user-defined types as values for <code>extra_info</code> .

### **CosTradingDynamic::DPEvalFailure Exception**

```
exception DPEvalFailure
{
    CosTrading::PropertyName name;
    TypeCode returned_type;
    any extra_info;
};
```

Evaluation of a dynamic property failed.

<code>name</code>	Name of the property to be evaluated
<code>returned_type</code>	Value type expected for the property
<code>extra_info</code>	Additional information used for property evaluation

# CosTradingDynamic:: DynamicPropEval Interface

```
interface DynamicPropEval
```

Defines a single operation for evaluating a dynamic property.

## DynamicPropEval::evalDP()

```
any evalDP(in CosTrading::PropertyName name,  
           in TypeCode returned_type,  
           in any extra_info)  
  raises(DPEvalFailure);
```

Evaluates a dynamic property and returns the objects properties.

### Parameters

<code>name</code>	Name of the property to be evaluated
<code>returned_type</code>	Value type expected for the property
<code>extra_info</code>	Additional information used for property evaluation

### Exceptions

<a href="#">DPEvalFailure</a>	Evaluation of the property failed
-------------------------------	-----------------------------------



# CosTradingRepos Module

Contains the `ServiceTypeRepository` interface, which manages information about service types for the trading service.

A service type represents the information needed to describe a service, including an interface type defining the computational signature of the service, and zero or more properties that augment the interface. Each traded service, or service offer, is associated with a service type.

There are several components of a service type:

**Interface:** The interface repository identifier for an interface determines the computational signature of a service. If the trading service is configured to use the interface repository, and this identifier resolves to an `InterfaceDef` object in the interface repository, then the trading service will ensure that an object in an exported offer conforms to this interface.

**Properties:** Any number of properties can be defined for a service type. Properties typically represent behavioral, non-functional and non-computational aspects of the service.

**Super types:** Service types can be related in a hierarchy that reflects interface type inheritance and property type aggregation. This hierarchy provides the basis for deciding if a service of one type may be substituted for a service of another type.

When a new service type is added that has one or more super types, the service type repository performs a number of consistency checks. First, the repository ensures (if possible) that the interface of the new type conforms to the interface of the super type. Second, the repository checks for any property that has been redefined in the new service type to ensure that it has the same type as that of the super type, and that its mode is at least as strong as its mode in the super type.



# CosTradingRepos::ServiceTypeRepository Interface

```
interface ServiceTypeRepository
```

Contains types and operations for managing the repository.

## ServiceTypeRepository::Identifier Alias

```
typedef CosTrading::Istring Identifier;
```

The interface repository identifier of an interface. For example, the identifier of this interface is `IDL:omg.org/CosTradingRepos/ServiceTypeRepository:1.0`.

## ServiceTypeRepository::PropStructSeq Sequence

```
typedef sequence<PropStruct> PropStructSeq;
```

## ServiceTypeRepository::ServiceTypeNameSeq Sequence

```
typedef sequence<CosTrading::ServiceTypeName> ServiceTypeNameSeq;
```

## ServiceTypeRepository::ListOption Enum

```
enum ListOption
{
    all,
    since
};
```

---

Indicates which service types are of interest.

all	All service types
since	All service types since a particular incarnation

## ServiceTypeRepository::PropertyMode Enum

```
enum PropertyMode
{
    PROP_NORMAL,
    PROP_READONLY,
    PROP_MANDATORY,
    PROP_MANDATORY_READONLY
};
```

Each property has a mode associated with it. The property mode places restrictions on an exporter when exporting and modifying service offers.

PROP_NORMAL	Property is optional
PROP_READONLY	Property is optional, but once a value has been supplied, it cannot be changed
PROP_MANDATORY	A value for this property must be supplied when the offer is exported, but can also be changed at some later time
PROP_MANDATORY_READONLY	A value for this property must be supplied when the offer is exported, and cannot be changed

## ServiceTypeRepository::IncarnationNumber Structure

```
struct IncarnationNumber
{
    unsigned long high;
    unsigned long low;
};
```

Represents a unique, 64-bit identifier that is assigned to each service type. This will be replaced by `long long` when that type is widely supported.



---

## ServiceTypeRepository::PropStruct Structure

```
struct PropStruct
{
    CosTrading::PropertyName name;
    TypeCode value_type;
    PropertyMode mode;
};
```

A complete description of a property.

name	Name of the property
value_type	CORBA::TypeCode describing the type of values allowed for the property
mode	Determines whether a property is mandatory, and whether the property can be modified

## ServiceTypeRepository::TypeStruct Structure

```
struct TypeStruct
{
    Identifier if_name;
    PropStructSeq props;
    ServiceTypeNameSeq super_types;
    boolean masked;
    IncarnationNumber incarnation;
};
```

A complete description of a service type.

if_name	Interface repository identifier for an interface
props	Defines the properties associated with this type
super_types	Service types from which this type inherits property definitions
masked	If TRUE, no new offers can be exported for this type
incarnation	Unique, 64-bit identifier for this type

---

## ServiceTypeRepository::SpecifiedServiceTypes Union

```
union SpecifiedServiceTypes switch(ListOption)
{
case since: IncarnationNumber incarnation;
};
```

Provides two ways of retrieving the names of the service types managed by the repository. The union's discriminator can be set to `all` if you want to obtain all of the service type names.

`since`                      Set this value with an incarnation number; only the names of those types whose incarnation numbers are greater than or equal to this value will be returned

## ServiceTypeRepository::AlreadyMasked Exception

```
exception AlreadyMasked {CosTrading::ServiceTypeName name};
```

The service type cannot be masked if it is already masked.

## ServiceTypeRepository::DuplicateServiceTypeName Exception

```
exception DuplicateServiceTypeName
{
CosTrading::ServiceTypeName name;
};
```

The same service type appeared more than once in the list of super types.

## ServiceTypeRepository::HasSubTypes Exception

```
exception HasSubTypes
{
CosTrading::ServiceTypeName the_type;
CosTrading::ServiceTypeName sub_type;
};
```

---

A service type cannot be removed if it is the super type of any other type.

### **ServiceTypeRepository::InterfaceTypeMismatch Exception**

```
exception InterfaceTypeMismatch
{
    CosTrading::ServiceTypeName base_service;
    Identifier base_if;
    CosTrading::ServiceTypeName derived_service;
    Identifier derived_if;
};
```

The interface of the new (*derived*) service type does not conform to the interface of a super type (*base service*).

### **ServiceTypeRepository::NotMasked Exception**

```
exception NotMasked {CosTrading::ServiceTypeName name};
```

The service type cannot be unmasked if it is not currently masked.

### **ServiceTypeRepository::ServiceTypeExists Exception**

```
exception ServiceTypeExists {CosTrading::ServiceTypeName name};
```

Another service type exists with the given name.

### **ServiceTypeRepository::ValueTypeRedefinition Exception**

```
exception ValueTypeRedefinition
{
    CosTrading::ServiceTypeName type_1;
    PropStruct definition_1;
    CosTrading::ServiceTypeName type_2;
    PropStruct definition_2;
};
```

---

The definition of a property in the new service type (*type\_1*) conflicts with the definition in a super type (*type\_2*). This error can result if the `value_type` members do not match, or if the mode of the property is weaker than in the super type.

## ServiceTypeRepository::incarnation Attribute

readonly attribute [IncarnationNumber](#) incarnation;

Determines the next incarnation number that will be assigned to a new service type. This could be used to synchronize two or more service type repositories, for example.

## ServiceTypeRepository::add\_type()

```
IncarnationNumber add_type(in CosTrading::ServiceTypeName name,  
                           in Identifier if_name,  
                           in PropStructSeq props,  
                           in ServiceTypeNameSeq super_types)  
raises(CosTrading::IllegalServiceType,  
       ServiceTypeExists,  
       InterfaceTypeMismatch,  
       CosTrading::IllegalPropertyName,  
       CosTrading::DuplicatePropertyName,  
       ValueTypeRedefinition,  
       CosTrading::UnknownServiceType,  
       DuplicateServiceTypeName);
```

Adds a new service type and returns a unique identifier for the new type.

### Parameters

<code>name</code>	Name to be used for the new type
<code>if_name</code>	Interface repository identifier for an interface
<code>props</code>	Properties defined for this interface interface
<code>super_types</code>	Zero or more super types from which this type will inherit interface and property definitions

---

## Exceptions

<code>CosTrading::</code> <a href="#">IllegalServiceType</a>	Service type name is empty or has an invalid format
<a href="#">ServiceTypeExists</a>	Service type already exists with the same name
<a href="#">InterfaceTypeMismatch</a>	Interface of the new type does not conform to the interface of a super type
<code>CosTrading::</code> <a href="#">IllegalPropertyName</a>	Property name is empty or has an invalid format
<code>CosTrading::</code> <a href="#">DuplicatePropertyName</a>	Same property name appears more than once in <code>props</code>
<a href="#">ValueTypeRedefinition</a>	Property definition in <code>props</code> conflicts with a definition in a super type
<code>CosTrading::</code> <a href="#">UnknownServiceType</a>	Super type does not exist
<a href="#">DuplicateServiceTypeName</a>	Same super type name appears more than once in <code>super_types</code>

## ServiceTypeRepository::describe\_type()

```
TypeStruct describe_type(in CosTrading::ServiceTypeName name)  
raises(CosTrading::IllegalServiceType,  
       CosTrading::UnknownServiceType);
```

Gets the description of a service type and returns a `TypeStruct` with the description.

## Parameters

`name` Name of the type of interest

## Exceptions

<code>CosTrading::</code> <a href="#">IllegalServiceType</a>	Service type name is empty or has an invalid format
<code>CosTrading::</code> <a href="#">UnknownServiceType</a>	Service type does not exist

---

## ServiceTypeRepository::fully\_describe\_type()

```
TypeStruct fully_describe_type(in CosTrading::ServiceTypeName
                                name)
raises(CosTrading::IllegalServiceType,
       CosTrading::UnknownServiceType);
```

Obtains the *full* description of a service type. The `super_types` member of a full description contains the names of the types in the transitive closure of the super type relation. The `props` member includes all properties inherited from the transitive closure of the super types. A [TypeStruct](#) containing the full description is returned.

### Parameters

`name`                    Name of the type of interest

### Exceptions

`CosTrading::`            Service type name is empty or has an invalid format  
[IllegalServiceType](#)

`CosTrading::`            Service type does not exist  
[UnknownServiceType](#)

## ServiceTypeRepository::list\_types()

```
ServiceTypeNameSeq list_types(in SpecifiedServiceTypes
                                which_types);
```

Lists the names of some or all of the service types in the repository.

### Parameters

`which_types`            Specifies which types are of interest

## ServiceTypeRepository::mask\_type()

```
void mask_type(in CosTrading::ServiceTypeName name)
raises(CosTrading::IllegalServiceType,
       CosTrading::UnknownServiceType,
       AlreadyMasked);
```

---

Masks a service type so that offers can no longer be exported for it. Masking a service type is useful when the type is considered deprecated; in other words, no new offers should be allowed, but existing offers are still supported.

### Parameters

`name` Name of the type to be masked

### Exceptions

`CosTrading::IllegalServiceType` Service type name is empty or has an invalid format

`CosTrading::UnknownServiceType` Service type does not exist

`AlreadyMasked` Service type is already masked

## ServiceTypeRepository::remove\_type()

```
void remove_type(in CosTrading::ServiceTypeName name)
raises(CosTrading::IllegalServiceType,
       CosTrading::UnknownServiceType,
       HasSubTypes);
```

Removes an existing service type.

### Parameters

`name` Name of the type to be removed

### Exceptions

`CosTrading::IllegalServiceType` Service type name is empty or has an invalid format

`CosTrading::UnknownServiceType` Service type does not exist

`HasSubTypes` Service type cannot be removed if it is the super type of any other type

---

## ServiceTypeRepository::unmask\_type()

```
void unmask_type(in CosTrading::ServiceTypeName name)
raises(CosTrading::IllegalServiceType,
       CosTrading::UnknownServiceType,
       NotMasked);
```

Unmasks a masked service type so that offers can be exported for it.

### Parameters

name                    Name of the type to be unmasked

### Exceptions

<a href="#">CosTrading::</a>	Service type name is empty or has an invalid format <a href="#">IllegalServiceType</a>
<a href="#">CosTrading::</a>	Service type does not exist <a href="#">UnknownServiceType</a>
<a href="#">NotMasked</a>	Service type is not currently masked



# CosTransactions Overview

The Object Management Group's (OMG) object transaction service (OTS) defines interfaces that integrate transactions into the distributed object paradigm. The OTS interface enables developers to manage transactions under two different models of transaction propagation, implicit and explicit:

- In the implicit model, the transaction context is associated with the client thread; when client requests are made on transactional objects, the transaction context associated with the thread is propagated to the object implicitly.
- In the explicit model, the transaction context must be passed explicitly when client requests are made on transactional objects in order to propagate the transaction context to the object.

Keep the following in mind:

- The `CosTransactions` Java classes are part of the package `org.omg.CosTransactions`.
- All of the OTS class methods can throw the `CORBA::SystemException` exception if an object request broker (ORB) error occurs.

## Overview of Classes

The OTS classes provide the following functionality:

- Managing transactions under the implicit model:  
[Current](#)
- Managing transactions under the explicit model:  
[TransactionFactory](#)  
[Control](#)  
[Coordinator](#)  
[Terminator](#)
- Managing resources in the CORBA environment:  
[RecoveryCoordinator](#)  
[Resource](#)

- 
- [SubtransactionAwareResource](#)  
[Synchronization](#)
  - Defining transactional interfaces in the CORBA environment:  
[TransactionalObject](#)
  - Reporting system errors:  
[HeuristicCommit](#)  
[HeuristicHazard](#)  
[HeuristicMixed](#)  
[HeuristicRollback](#)  
[Inactive](#)  
[InvalidControl](#)  
INVALID\_TRANSACTION  
[NoTransaction](#)  
[NotPrepared](#)  
[NotSubtransaction](#)  
[SubtransactionsUnavailable](#)  
TRANSACTION\_MODE  
TRANSACTION\_REQUIRED  
TRANSACTION\_ROLLEDBACK  
TRANSACTION\_UNAVAILABLE  
[Unavailable](#)

## General Exceptions

Errors are handled in OTS by using exceptions. Exceptions provide a way of returning error information back through multiple levels of procedure or method calls, propagating this information until a method or procedure is reached that can respond appropriately to the error.

Each of the following exceptions are implemented as classes. The exceptions are shown here in two tables: one for the OTS exceptions and another for the system exceptions.

**Table 8:** *OTS Exceptions*

Exception	Description
HeuristicCommit	This exception is thrown to report that a heuristic decision was made by one or more participants in a transaction and that all updates have been committed. See Also:  <a href="#">Resource</a> class
HeuristicHazard	This exception is thrown to report that a heuristic decision has possibly been made by one or more participants in a transaction and the outcome of all participants in the transaction is unknown. See Also:  <a href="#">Current::commit()</a> <a href="#">Resource</a> class <a href="#">Terminator::commit()</a>
HeuristicMixed	This exception is thrown to report that a heuristic decision was made by one or more participants in a transaction and that some updates have been committed and others rolled back. See Also:  <a href="#">Current::commit()</a> <a href="#">Resource</a> class <a href="#">Terminator::commit()</a>
HeuristicRollback	This exception is thrown to report that a heuristic decision was made by one or more participants in a transaction and that all updates have been rolled back. See Also:  <a href="#">Resource</a> class

---

**Table 8:** *OTS Exceptions*

---

<b>Exception</b>	<b>Description</b>
Inactive	This exception is thrown when a transactional operation is requested for a transaction, but that transaction is already prepared. See Also: <a href="#">Coordinator::create_subtransaction()</a> <a href="#">Coordinator::register_resource()</a> <a href="#">Coordinator::register_subtran_aware()</a> <a href="#">Coordinator::rollback_only()</a>
InvalidControl	This exception is thrown when an invalid <a href="#">Control</a> object is used in an attempt to resume a suspended transaction. See Also: <a href="#">Control</a> class <a href="#">Current::resume()</a>
NotPrepared	This exception is thrown when an operation (such as a <a href="#">commit()</a> ) is requested for a resource, but that resource is not prepared. See Also: <a href="#">RecoveryCoordinator::replay_completion()</a> <a href="#">Resource</a> class
NoTransaction	This exception is thrown when an operation is requested for the current transaction, but no transaction is associated with the client thread. See Also: <a href="#">Current::commit()</a> <a href="#">Current::rollback()</a> <a href="#">Current::rollback_only()</a>
NotSubtransaction	This exception is thrown when an operation that requires a subtransaction is requested for a transaction that is not a subtransaction. See Also: <a href="#">Coordinator::register_subtran_aware()</a>
SubtransactionsUnavailable	This exception is thrown when an attempt is made to create a subtransaction. See Also: <a href="#">Coordinator::create_subtransaction()</a> <a href="#">Current::begin()</a>

---

**Table 8:** *OTS Exceptions*

Exception	Description
Unavailable	This exception is thrown when a <a href="#">Terminator</a> or <a href="#">Coordinator</a> object cannot be provided by a <a href="#">Control</a> object due to environment restrictions. See Also: <a href="#">Control::get_coordinator()</a> <a href="#">Control::get_terminator()</a>

The following table shows the system exceptions that can be thrown:

**Table 9:** *System Exceptions*

Exception	Description
INVALID_TRANSACTION	This exception is raised when the transaction context is invalid for a request.
TRANSACTION_MODE	This exception is raised when there is a mismatch between the transaction policy in the target object's IOR and the current transaction mode (see <a href="#">Table 1</a> ).
TRANSACTION_REQUIRED	This exception is raised when an invocation on an object expecting a transaction is performed with no transaction (see <a href="#">Table 1</a> ).
TRANSACTION_ROLLEDBACK	This exception is raised when a transactional operation (such as <code>commit()</code> ) is requested for a transaction that has been rolled back or marked for rollback. See Also: <a href="#">Current::commit()</a> <a href="#">Terminator::commit()</a>
TRANSACTION_UNAVAILABLE	This exception is raised when a transaction invocation is requested but the transaction service is not available.

---

## General Data Types

OTS defines enumerated data types to represent the status of a transaction object during its lifetime and to indicate a participant's vote on the outcome of a transaction.

### Status Enumeration Type

```
enum Status{
    StatusActive,
    StatusMarkedRollback,
    StatusPrepared,
    StatusCommitted,
    StatusRolledBack,
    StatusUnknown,
    StatusNoTransaction,
    StatusPreparing,
    StatusCommitting,
    StatusRollingBack
};
```

The `Status` enumerated type defines values that are used to indicate the status of a transaction. Status values are used in both the implicit and explicit models of transaction demarcation defined by OTS. The [Current::get\\_status\(\)](#) operation can be called to return the transaction status if the implicit model is used. The [Coordinator::get\\_status\(\)](#) operation can be called to return the transaction status if the explicit model is used.

The `Status` values indicate the following:

<code>StatusActive</code>	Processing of a transaction is still in progress.
<code>StatusMarkedRollback</code>	A transaction is marked to be rolled back.
<code>StatusPrepared</code>	A transaction has been prepared but not completed.
<code>StatusCommitted</code>	A transaction has been committed and the effects of the transaction have been made permanent.

---

<code>StatusActive</code>	Processing of a transaction is still in progress.
<code>StatusRolledBack</code>	A transaction has been rolled back.
<code>StatusUnknown</code>	The status of a transaction is unknown.
<code>StatusNoTransaction</code>	A transaction does not exist in the current transaction context.
<code>StatusPreparing</code>	A transaction is preparing to commit.
<code>StatusCommitting</code>	A transaction is in the process of committing.
<code>StatusRollingBack</code>	A transaction is in the process of rolling back.

**See Also**

[CosTransactions::Coordinator::get\\_status\(\)](#)  
[CosTransactions::Current::get\\_status\(\)](#)

**Vote Enumeration Type**

```
enum Vote{
    VoteCommit,
    VoteRollback,
    VoteReadOnly
};
```

The `Vote` enumerated type defines values for the voting status of transaction participants. The participants in a transaction each vote on the outcome of a transaction during the two-phase commit process. In the prepare phase, a `Resource` object can vote whether to commit or abort a transaction. If a `Resource` has not modified any data as part of the transaction, it can vote `VoteReadOnly` to indicate that its participation does not affect the outcome of the transaction. The `Vote` values specify the following:

<code>VoteCommit</code>	The value used to indicate a vote to commit a transaction.
<code>VoteRollback</code>	The value used to indicate a vote to abort (rollback) a transaction.
<code>VoteReadOnly</code>	The value used to indicate no vote on the outcome of a transaction.

---

See Also [CosTransactions::Resource](#)

## OTSPolicyValue Data Type

```
typedef unsigned short OTSPolicyValue;  
const OTSPolicyValue REQUIRES = 1;  
const OTSPolicyValue FORBIDS = 2;  
const OTSPolicyValue ADAPTS = 3;  
const CORBA::PolicyType OTS_POLICY_TYPE = 56;
```

The `OTSPolicyValue` data type is used to create POA policy objects that define behavior of objects during invocations, both with and without a current transaction.

The `CORBA::ORB::create_policy()` operation is used to create the policy objects (passing in the appropriate `OTSPolicyValue` value). The policy object is passed in the list of policy objects passed to `PortableServer::POA::create_POA()`.

The `OTSPolicyValue` values indicate the following:

REQUIRES	The target object depends on the presence of a transaction. If there is no current transaction, a <a href="#">TRANSACTION_REQUIRED</a> system exception is raised.
FORBIDS	The target object depends on the absence of a transaction. If there is a current transaction, the <a href="#">INVALID_TRANSACTION</a> system exception is raised. When there is no current transaction, the behavior of the <code>FORBIDS</code> policy is also affected by the <code>NonTxTargetPolicy</code> .
ADAPTS	The target object is invoked within the current transaction, whether there is one or not.

You cannot create a POA that mixes the `OTSPolicyValue` `FORBIDS` or `ADAPTS` values with the [InvocationPolicyValue](#) `EITHER` or `UNSHARED` values. Attempting to do so raises `PortableServer::InvalidPolicy` exception.

### Examples

The following example shows the `ADAPTS` value:

```
//Java  
ORB orb = ...  
Any policy_val = orb.create_any();
```



```
OTSPolicyValueHelper.insert(policy_val, ADAPTS.value);
Policy tx_policy =
    orb.create_policy(OTS_POLICY_TYPE.value,
                    policy_val);
```

**See Also**

[CosTransactions::NonTxTargetPolicyValue](#)  
[CosTransactions::TransactionalObject](#)

## InvocationPolicyValue Data Type

```
typedef unsigned short InvocationPolicyValue;
const InvocationPolicyValue EITHER = 0;
const InvocationPolicyValue SHARED = 1;
const InvocationPolicyValue UNSHARED = 2;
const CORBA::PolicyType INVOCATION_POLICY_TYPE = 55;
```

The `InvocationPolicyValue` data type is used to create POA policy objects that define the behavior of objects with respect to the *shared* and *unshared* transaction models.

The shared transaction model represents a standard end-to-end transaction that is shared between the client and the target object. The unshared transaction model uses asynchronous messaging where separate transactions are used along the invocation path. Hence, the client and the target object do not share the same transaction.

The `CORBA::ORB::create_policy()` operation is used to create the policy objects (passing in the appropriate `InvocationPolicyValue`). The policy object is passed in the list of policy objects passed to `PortableServer::POA::create_POA()`.

The `InvocationPolicyValue` data type values indicate the following:

EITHER	The target object supports both shared and unshared invocations.
SHARED	The target object supports synchronous invocations and asynchronous includes that do not involve a routing element.
UNSHARED	The target object.

---

You cannot create a POA that mixes the `InvocationPolicyValue` EITHER OR UNSHARED values with the [OTSPolicyValue FORBIDS](#) or [ADAPTS](#) values. Attempting to do this raises a [PortableServer::InvalidPolicy](#) exception.

If no `InvocationPolicy` object is passed to [create\\_POA\(\)](#), the `InvocationPolicy` defaults to SHARED.

---

**Note:** The unshared transaction model is not supported in this release.

---

## Examples

The following example shows the SHARED value:

```
//Java
ORB orb = ...
Any policy_val = orb.create_any();
InvocationPolicyValueHelper.insert(policy_val, SHARED.value);
Policy tx_policy =
    orb.create_policy(INVOCATION_POLICY_TYPE.value,
                    policy_val);
```

## See Also

[CosTransactions::OTSPolicyValue](#)  
[CosTransactions::NonTxTargetPolicyValue](#)

## NonTxTargetPolicyValue Data Type

```
typedef unsigned short NonTxTargetPolicyValue;
const NonTxTargetPolicyValue PREVENT = 0;
const NonTxTargetPolicyValue PERMIT = 1;
const CORBA::PolicyType NON_TX_TARGET_POLICY_TYPE = 57;
```

The `NonTxTargetPolicyValue` data type is used to create policy objects used by clients to affect the behavior of invocations on objects with an `OTSPolicy` of FORBIDS.

The [CORBA::ORB::create\\_policy\(\)](#) operation creates the policy objects (passing the appropriate `NonTxTargetPolicyValue`). The policy object is passed in the list of policy objects passed to [CORBA::PolicyManager::set\\_policy\\_overrides\(\)](#) and [CORBA::PolicyCurrent::set\\_policy\\_overrides\(\)](#).

See the [CORBA::PolicyCurrent](#) and [CORBA::PolicyManager](#) classes for more details on setting policies.

The behavior of the `NonTxTargetPolicy` values apply to invocations where there is a current transaction and the target object has the [OTSPolicyValue](#) of [FORBIDS](#). The `NonTxTargetPolicy` values indicate the following:

PREVENT	The invocation is prevented from proceeding and the system exception <a href="#">INVALID_TRANSACTION</a> is raised.
PERMIT	The invocation proceeds but not in the context of the current transaction.

The default `NonTxTargetPolicy` is `PREVENT`.

### Examples

The following example shows the `PERMIT` value:

```
//Java
ORB orb = ...
Any policy_val = orb.create_any();
NonTxTargetPolicyValueHelper.insert(policy_val, PERMIT.value);
Policy tx_policy =
    orb.create_policy(NON_TX_TARGET_POLICY_TYPE.value,
                    policy_val);
```

### See Also

[CosTransactions::OTSPolicyValue](#)  
[CosTransactions::InvocationPolicyValue](#)

## TransactionPolicyValue Data Type

```
typedef unsigned short TransactionPolicyValue;
const TransactionPolicyValue Allows_shared = 0;
const TransactionPolicyValue Allows_none = 1;
const TransactionPolicyValue Requires_shared = 2;
const TransactionPolicyValue Allows_unshared = 3;
const TransactionPolicyValue Allows_either = 4;
const TransactionPolicyValue Requires_unshared = 5;
const TransactionPolicyValue Requires_either = 6;
const CORBA::PolicyType TRANSACTION_POLICY_TYPE = 36;
```

The `TransactionalPolicyValue` data type has been deprecated and replaced with the [OTSPolicyValue](#) and [InvocationPolicyValue](#) types.

The `TransactionalPolicyValue` data type has been retained in this release for backward compatibility. See the *CORBA Programmer's Guide* for details of interoperability with previous Orbix releases.



# CosTransactions::Control Class

The `Control` class enables explicit control of a factory-created transaction; the factory creates a transaction and returns a `Control` instance associated with the transaction. The `Control` object provides access to the [Coordinator](#) and [Terminator](#) objects used to manage and complete the transaction.

A `Control` object can be used to propagate a transaction context explicitly. By passing a `Control` object as an argument in a request, the transaction context can be propagated. [TransactionFactory::create\(\)](#) can be used to create a transaction and return the `Control` object associated with it.

```
// Java
public interface Control
    extends ControlOperations,
        org.omg.CORBA.Object,
        org.omg.CORBA.portable.IDLEntity
{
}
```

The `Control` class extends `ControlOperations`:

```
public interface ControlOperations
{
    org.omg.CosTransactions.Terminator get_terminator() throws
    org.omg.CosTransactions.Unavailable;

    org.omg.CosTransactions.Coordinator get_coordinator() throws
    org.omg.CosTransactions.Unavailable;
}
```

## See Also

[CosTransactions::Coordinator](#)  
[CosTransactions::Current::get\\_control\(\)](#)  
[CosTransactions::Coordinator::get\\_status\(\)](#)  
[CosTransactions::Terminator](#)  
[CosTransactions::TransactionFactory::create\(\)](#)  
[NoTransaction](#)  
[NotSubtransaction](#)

---

## Control::get\_coordinator()

```
// Java
org.omg.CosTransactions.Coordinator get_coordinator() throws
    org.omg.CosTransactions.Unavailable;
```

`get_coordinator()` returns the [Coordinator](#) object for the transaction with which the `Control` object is associated. The returned [Coordinator](#) object can be used to determine the status of the transaction, the relationship between the associated transaction and other transactions, to create subtransactions, and so on.

### Exceptions

[Unavailable](#) The [Coordinator](#) associated with the `Control` object is not available.

### See Also

[CosTransactions::Coordinator](#)

## Control::get\_terminator()

```
// Java
org.omg.CosTransactions.Terminator get_terminator() throws
    org.omg.CosTransactions.Unavailable;
```

`get_terminator()` returns the [Terminator](#) object for the transaction with which the `Control` object is associated. The returned [Terminator](#) object can be used to either commit or roll back the transaction.

### Exceptions

[Unavailable](#) The [Terminator](#) associated with the `Control` object is not available.

### See Also

[CosTransactions::Terminator](#)

# CosTransactions::Coordinator Class

The `Coordinator` class enables explicit control of a factory-created transaction; the factory creates a transaction and returns a [Control](#) instance associated with the transaction. [Control::get\\_coordinator\(\)](#) returns the `Coordinator` object used to manage the transaction.

The operations defined by the `Coordinator` class can be used by the participants in a transaction to determine the status of the transaction, determine the relationship of the transaction to other transactions, mark the transaction for rollback, and create subtransactions.

The `Coordinator` class also defines operations for registering resources as participants in a transaction and registering subtransaction-aware resources with a subtransaction.

```
// Java
package org.omg.CosTransactions;

public interface Coordinator
    extends CoordinatorOperations,
        org.omg.CORBA.Object,
        org.omg.CORBA.portable.IDLEntity
{
}
```

The `Coordinator` class extends `CoordinatorOperations`:

```
public interface CoordinatorOperations
{
    org.omg.CosTransactions.Status get_status();

    org.omg.CosTransactions.Status get_parent_status();

    org.omg.CosTransactions.Status get_top_level_status();

    boolean is_same_transaction(
        org.omg.CosTransactions.Coordinator tc
    );
}
```

---

```
boolean is_related_transaction(
    org.omg.CosTransactions.Coordinator tc
);

boolean is_ancestor_transaction(
    org.omg.CosTransactions.Coordinator tc
);

boolean is_descendant_transaction(
    org.omg.CosTransactions.Coordinator tc
);

boolean is_top_level_transaction();

int hash_transaction();

int hash_top_level_tran();

org.omg.CosTransactions.RecoveryCoordinator
register_resource(
    org.omg.CosTransactions.Resource r
) throws org.omg.CosTransactions.Inactive;

void register_synchronization(
    org.omg.CosTransactions.Synchronization sync
) throws org.omg.CosTransactions.Inactive,
org.omg.CosTransactions.SynchronizationUnavailable;

void register_subtran_aware(
    org.omg.CosTransactions.SubtransactionAwareResource r
) throws org.omg.CosTransactions.Inactive,
org.omg.CosTransactions.NotSubtransaction;

void rollback_only() throws org.omg.CosTransactions.Inactive;

java.lang.String get_transaction_name();

org.omg.CosTransactions.Control create_subtransaction()
throws org.omg.CosTransactions.SubtransactionsUnavailable,
org.omg.CosTransactions.Inactive;

org.omg.CosTransactions.PropagationContext get_txcontext()
throws org.omg.CosTransactions.Unavailable;
```



---

**See Also**

```
}  
CosTransactions::Control  
CosTransactions::Control::get\_coordinator\(\)  
CosTransactions::Terminator
```

**Coordinator::create\_subtransaction()**

```
// Java  
org.omg.CosTransactions.Control create\_subtransaction\(\) throws  
org.omg.CosTransactions.SubtransactionsUnavailable,  
org.omg.CosTransactions.Inactive;
```

`create_subtransaction()` returns the `Control` object associated with the new subtransaction.

`create_subtransaction()` creates a new subtransaction for the transaction associated with the `Coordinator` object. A subtransaction is one that is embedded within another transaction; the transaction within which the subtransaction is embedded is referred to as its parent. A transaction that has no parent is a top-level transaction. A subtransaction executes within the scope of its parent transaction and can be used to isolate failures; if a subtransaction fails, only the subtransaction is rolled back. If a subtransaction commits, the effects of the commit are not permanent until the parent transaction commits. If the parent transaction rolls back, the subtransaction is also rolled back.

**Exceptions**

[Subtransaction](#) Subtransactions are not supported.  
[sUnavailable](#)  
[e](#)

[Inactive](#) The transaction is already prepared.

**See Also**

[CosTransactions::Control](#)

**Coordinator::get\_parent\_status()**

```
// Java  
org.omg.CosTransactions.Status get\_parent\_status\(\);
```

---

`get_parent_status()` returns the status of the parent of the transaction associated with the `Coordinator` object. For more information, see [create\\_subtransaction\(\)](#).

The status returned indicates which phase of processing the transaction is in. See the reference page for the [Status](#) type for information about the possible status values. If the transaction associated with the `Coordinator` object is a subtransaction, the status of its parent transaction is returned. If there is no parent transaction, the status of the transaction associated with the `Coordinator` object itself is returned.

**See Also**

[CosTransactions::Coordinator::create\\_subtransaction\(\)](#)  
[CosTransactions::Coordinator::get\\_status\(\)](#)  
[CosTransactions::Coordinator::get\\_top\\_level\\_status\(\)](#)  
[CosTransactions::Status](#)

## **Coordinator::get\_status()**

```
// Java  
org.omg.CosTransactions.Status get_status();
```

`get_status()` returns the status of the transaction associated with the `Coordinator` object. The status returned indicates which phase of processing the transaction is in. See the reference page for the [Status](#) type for information about the possible status values.

**See Also**

[CosTransactions::Coordinator::get\\_parent\\_status\(\)](#)  
[CosTransactions::Coordinator::get\\_top\\_level\\_status\(\)](#)  
[CosTransactions::Status](#)

## **Coordinator::get\_top\_level\_status()**

```
// Java  
org.omg.CosTransactions.Status get_top_level_status();
```

`get_top_level_status()` returns the status of the top-level ancestor of the transaction associated with the `Coordinator` object. See [Coordinator::create\\_subtransaction\(\)](#) for more information.

---

The status returned indicates which phase of processing the transaction is in. See the reference page for the [Status](#) type for information about the possible status values. If the transaction associated with the `Coordinator` object is the top-level transaction, its status is returned.

#### See Also

[CosTransactions::Coordinator::create\\_subtransaction\(\)](#)  
[CosTransactions::Coordinator::get\\_status\(\)](#)  
[CosTransactions::Coordinator::get\\_parent\\_status\(\)](#)  
[CosTransactions::Status](#)

### Coordinator::get\_transaction\_name()

```
// Java
java.lang.String get_transaction_name();
```

`get_transaction_name()` returns the name of the transaction associated with the `Coordinator` object.

### Coordinator::get\_txcontext()

```
// Java
org.omg.CosTransactions.PropagationContext get_txcontext() throws
    org.omg.CosTransactions.Unavailable;
```

Returns the propagation context object which is used to export the current transaction to a new transaction service domain.

#### Exceptions

[Unavailable](#) The propagation context is unavailable.

#### See Also

[CosTransactions::TransactionFactory::recreate\(\)](#)

### Coordinator::hash\_top\_level\_tran()

```
// Java
int hash_top_level_tran();
```

`hash_top_level_tran()` returns a hash code for the top-level ancestor of the transaction associated with the `Coordinator` object. If the transaction associated with the `Coordinator` object is the top-level transaction, its hash code is

---

returned. See [create\\_subtransaction\(\)](#) for more information. The returned hash code is typically used as an index into a table of `Coordinator` objects. The low-order bits of the hash code can be used to hash into a table with a size that is a power of two.

**See Also**

[CosTransactions::Coordinator::create\\_subtransaction\(\)](#)  
[CosTransactions::Coordinator::hash\\_transaction\(\)](#)

## **Coordinator::hash\_transaction()**

```
// Java
int hash_transaction();
```

`hash_transaction()` returns a hash code for the transaction associated with the `Coordinator` object.

**See Also**

[CosTransactions::Coordinator::hash\\_top\\_level\\_tran\(\)](#)

## **Coordinator::is\_ancestor\_transaction()**

```
// Java
boolean is_ancestor_transaction(
    org.omg.CosTransactions.Coordinator tc
);
```

`is_ancestor_transaction()` returns true if the transaction is an ancestor or if the two transactions are the same; otherwise, the method returns false.

**Parameters**

`tc` Specifies the coordinator of another transaction to compare with the `Coordinator` object.

`is_ancestor_transaction()` determines whether the transaction associated with the `Coordinator` object is an ancestor of the transaction associated with the coordinator specified in the `tc` parameter. See [create\\_subtransaction\(\)](#) for more information.

**See Also**

[CosTransactions::Coordinator::is\\_descendant\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_related\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_same\\_transaction\(\)](#)  
[CosTransactions::Coordinator::create\\_subtransaction\(\)](#)

---

## Coordinator::is\_descendant\_transaction()

```
// Java
boolean is_descendant_transaction(
    org.omg.CosTransactions.Coordinator tc
);
```

`is_descendant_transaction()` returns true if the transaction is a descendant or if the two transactions are the same; otherwise, the method returns false.

### Parameters

`tc` Specifies the coordinator of another transaction to compare with the `Coordinator` object.

`is_descendant_transaction()` determines whether the transaction associated with the `Coordinator` object is a descendant of the transaction associated with the coordinator specified in the `tc` parameter. See [Coordinator::create\\_subtransaction\(\)](#) for more information.

### See Also

[CosTransactions::Coordinator::is\\_descendant\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_related\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_same\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_top\\_level\\_transaction\(\)](#)  
[CosTransactions::Coordinator::create\\_subtransaction\(\)](#)

## Coordinator::is\_related\_transaction()

```
// Java
boolean is_related_transaction(
    org.omg.CosTransactions.Coordinator tc
);
```

`is_related_transaction()` returns true if both transactions are descendants of the same transaction; otherwise, the method returns false.

### Parameters

`tc` Specifies the coordinator of another transaction to compare with the `Coordinator` object.

---

`is_related_transaction()` determines whether the transaction associated with the `Coordinator` object and the transaction associated with the coordinator specified in the `tc` parameter have a common ancestor. See [create\\_subtransaction\(\)](#) for more information.

**See Also**

[CosTransactions::Coordinator::is\\_descendant\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_ancestor\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_same\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_top\\_level\\_transaction\(\)](#)  
[CosTransactions::Coordinator::create\\_subtransaction\(\)](#)

### **Coordinator::is\_same\_transaction()**

```
// Java
boolean is_same_transaction(
    org.omg.CosTransactions.Coordinator tc
);
```

`is_same_transaction()` returns true if the transactions associated with the two `Coordinator` objects are the same transaction; otherwise, the method returns false.

**Parameters**

`tc` Specifies the coordinator of another transaction to compare with the `Coordinator` object.

`is_same_transaction()` determines whether the transaction associated with the `Coordinator` object and the transaction associated with the coordinator specified in the `tc` parameter are the same transaction.

**See Also**

[CosTransactions::Coordinator::is\\_descendant\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_related\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_ancestor\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_top\\_level\\_transaction\(\)](#)

### **is\_top\_level\_transaction()**

```
// Java
boolean is_top_level_transaction();
```

---

`is_top_level_transaction()` returns true if the transaction is a top-level transaction; otherwise, the method returns false.  
`is_top_level_transaction()` determines whether the transaction associated with a `Coordinator` object is a top-level transaction. See [create\\_subtransaction\(\)](#) for more information.

#### See Also

[CosTransactions::Coordinator::is\\_descendant\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_related\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_same\\_transaction\(\)](#)  
[CosTransactions::Coordinator::is\\_ancestor\\_transaction\(\)](#)  
[CosTransactions::Coordinator::create\\_subtransaction\(\)](#)

### register\_resource()

```
// Java
org.omg.CosTransactions.RecoveryCoordinator register_resource(
    org.omg.CosTransactions.Resource r
    ) throws org.omg.CosTransactions.Inactive;
```

`register_resource()` registers a specified resource as a participant in the transaction associated with a `Coordinator` object. When the transaction ends, the registered resource must commit or roll back changes made as part of the transaction. Only server applications can register resources. See [Resource](#) class for more information. `register_resource()` returns a [RecoveryCoordinator](#) object that the registered [Resource](#) object can use during recovery.

#### Parameters

`resource`            The resource to register as a participant.

#### Exceptions

CORBA::TRANSACTION  
\_ROLLEDBACK            The transaction is marked for rollback only.

#### See Also

[CosTransactions::RecoveryCoordinator](#)  
[CosTransactions::Resource](#)

### register\_subtran\_aware()

```
// Java
void register_subtran_aware(
```

---

```
    org.omg.CosTransactions.SubtransactionAwareResource r
    ) throws org.omg.CosTransactions.Inactive,
    org.omg.CosTransactions.NotSubtransaction;
```

`register_subtran_aware()` registers a specified resource with the subtransaction associated with a `Coordinator` object. The resource is registered with the subtransaction only, not as a participant in the top-level transaction. ([register\\_resource\(\)](#) can be used to register the resource as a participant in the top-level transaction.) Only server applications can register resources.

## Parameters

`resource`            The resource to register.

When the transaction ends, the registered resource must commit or roll back changes made as part of the subtransaction. See the reference page for the [SubtransactionAwareResource](#) class for more information.

## Exceptions

[NotSubtransaction](#) The transaction associated with the `Coordinator` object is not a subtransaction

[Inactive](#)            The subtransaction or any ancestor of the subtransaction has ended.

`CORBA::TRANSACTION  
_ROLLEDBACK`        The transaction is marked for rollback only.

## See Also

[CosTransactions::RecoveryCoordinator](#)  
[CosTransactions::SubtransactionAwareResource](#)

## register\_synchronization()

```
// Java
void register_synchronization(
    org.omg.CosTransactions.Synchronization sync
    ) throws org.omg.CosTransactions.Inactive,
    org.omg.CosTransactions.SynchronizationUnavailable;
```

`register_synchronization()` registers a specified synchronization object for the transaction associated with a `Coordinator` object. See the reference page for the [Synchronization](#) class for more information.



---

## Parameters

`sync`            The synchronization object to register.

## Exceptions

[Inactive](#)            The transaction is already prepared.  
CORBA::  
    TRANSACTION  
    \_ROLLEDBACK        The transaction is marked for rollback only.

## See Also

[CosTransactions::RecoveryCoordinator](#)  
[CosTransactions::Synchronization](#)

## rollback\_only()

```
// Java  
void rollback_only() throws org.omg.CosTransactions.Inactive;
```

`rollback_only()` marks the transaction associated with the `Coordinator` object so that the only possible outcome for the transaction is to roll back. The transaction is not rolled back until the participant that created the transaction either commits or aborts the transaction.

OTS allows [Terminator::rollback\(\)](#) to be called instead of `rollback_only()`. Calling [Terminator::rollback\(\)](#) rolls back the transaction immediately, preventing unnecessary work from being done between the time the transaction is marked for rollback and the time the transaction is actually rolled back.

## Exceptions

[Inactive](#)            The transaction is already prepared.

## See Also

[CosTransactions::Terminator::rollback\(\)](#)



# CosTransactions::Current Class

The `Current` class represents a transaction that is associated with the calling thread; the thread defines the transaction context. The transaction context is propagated implicitly when the client issues requests.

This class defines member methods for beginning, committing, and aborting a transaction using the implicit model of transaction control. It also defines member methods for suspending and resuming a transaction and retrieving information about a transaction.

```
// Java
package org.omg.CosTransactions;

public interface Current extends
    org.omg.CORBA.Current
{
    void begin() throws
        org.omg.CosTransactions.SubtransactionsUnavailable;

    void commit(
        boolean report_heuristics
    ) throws org.omg.CosTransactions.NoTransaction,
        org.omg.CosTransactions.HeuristicMixed,
        org.omg.CosTransactions.HeuristicHazard;

    void rollback() throws org.omg.CosTransactions.NoTransaction;

    void rollback_only() throws
        org.omg.CosTransactions.NoTransaction;

    org.omg.CosTransactions.Status get_status();

    java.lang.String get_transaction_name();

    void set_timeout(
        int seconds);
}
```

---

```
int get_timeout();

org.omg.CosTransactions.Control get_control();

org.omg.CosTransactions.Control suspend();

void resume(
    org.omg.CosTransactions.Control which
) throws org.omg.CosTransactions.InvalidControl;

}
```

**See Also**

[CosTransactions::Control](#)  
[CosTransactions::Status](#)

### **Current::begin()**

```
// Java
void begin() throws
    org.omg.CosTransactions.SubtransactionsUnavailable;
```

`begin()` creates a new transaction and modifies the transaction context of the calling thread to associate the thread with the new transaction. If subtransactions are not available, an attempt to create a nested transaction throws the `SubtransactionsUnavailable` exception.

**See Also**

[CosTransactions::Current::commit\(\)](#)  
[CosTransactions::Current::rollback\(\)](#)  
[CosTransactions::Current::rollback\\_only\(\)](#)

### **Current::commit()**

```
// Java
void commit(
    boolean report_heuristics
) throws org.omg.CosTransactions.NoTransaction,
    org.omg.CosTransactions.HeuristicMixed,
    org.omg.CosTransactions.HeuristicHazard;
```

`commit()` attempts to commit the transaction associated with the calling thread.

---

## Parameters

`report_heuristics` specifies whether to report heuristic decisions for the transaction associated with the calling thread.

## Exceptions

`NoTransactionException` No transaction is associated with the calling thread.

[HeuristicMixed](#) The `report_heuristics` parameter is true and a heuristic decision causes inconsistent outcomes

[HeuristicHazard](#) The `report_heuristics` parameter is true and a heuristic decision might have caused inconsistent outcomes.

`TRANSACTION_ROLLEDBACK` Not all the transaction participants commit.

## See Also

[CosTransactions::Current::begin\(\)](#)  
[CosTransactions::Current::rollback\(\)](#)  
[CosTransactions::Current::rollback\\_only\(\)](#)

## Current::get\_control()

```
// Java
org.omg.CosTransactions.Control get_control();
```

`get_control()` returns the `Control` object for the transaction associated with the calling thread. If no transaction is associated with the calling thread, a null object reference is returned.

## See Also

[CosTransactions::Current::resume\(\)](#)

## Current::get\_status()

```
// Java
org.omg.CosTransactions.Status get_status();
```

`get_status()` returns the status of the transaction associated with the calling thread. If no transaction is associated with the calling thread, the `StatusNoTransaction` value is returned.

---

The status returned indicates the processing phase of the transaction. See the [Status](#) type for information about the possible status values.

**See Also** [CosTransactions::Status](#) Enumeration Type

## Current::get\_timeout()

```
// Java
int get_timeout()
```

Returns the timeout in seconds for transactions created using the [begin\(\)](#) operation.

**See Also** [CosTransactions::Current](#)  
[CosTransactions::Current::begin\(\)](#)  
[CosTransactions::Current::set\\_timeout\(\)](#)

## Current::get\_transaction\_name()

```
// Java
java.lang.String get_transaction_name();
```

`get_transaction_name()` returns the name of the transaction associated with the calling thread. If no transaction is associated with the calling thread, a null string is returned.

**See Also** [CosTransactions::Current](#)

## Current::resume()

```
// Java
void resume(
    org.omg.CosTransactions.Control which
) throws org.omg.CosTransactions.InvalidControl;
```

`resume()` resumes the suspended transaction identified by the `which` parameter and associated with the calling thread. If the value of the `which` parameter is a null object reference, the calling thread disassociates from the transaction. If the control object is invalid, the [InvalidControl](#) exception is thrown.

---

## Parameters

which Specifies a [Control](#) object that represents the transaction context associated with the calling thread.

## See Also

[CosTransactions::Current](#)  
[CosTransactions::Current::get\\_control\(\)](#)  
[CosTransactions::Current::suspend\(\)](#)

## Current::rollback()

```
// Java  
void rollback() throws org.omg.CosTransactions.NoTransaction;
```

`rollback()` rolls back the transaction associated with the calling thread. If the transaction was started with [begin\(\)](#), the transaction context for the thread is restored to its state before the transaction was started; otherwise, the transaction context is set to null.

## Exceptions

[NoTransaction](#) No transaction is associated with the calling thread.

## See Also

[CosTransactions::Current](#)  
[CosTransactions::Current::begin\(\)](#)  
[CosTransactions::Current::rollback\\_only\(\)](#)

## Current::rollback\_only()

```
// Java  
void rollback_only() throws org.omg.CosTransactions.NoTransaction;
```

`rollback_only()` marks the transaction associated with the calling thread for rollback. The transaction is modified so that the only possible outcome is to roll back the transaction. Any participant in the transaction can mark the transaction for rollback. The transaction is not rolled back until the participant that created the transaction either commits or aborts the transaction.

OTS allows [Current::rollback\(\)](#) to be called instead of `rollback_only()`. Calling [Current::rollback\(\)](#) rolls back the transaction immediately, preventing unnecessary work from being done between the time the transaction is marked for rollback and the time the transaction is actually rolled back.

---

## Exceptions

[NoTransaction](#) No transaction is associated with the calling thread.

## See Also

[CosTransactions::Current](#)  
[CosTransactions::Current::rollback\(\)](#)

## Current::set\_timeout()

```
// Java
void set_timeout(
    int seconds
);
```

`set_timeout()` sets a timeout period for the transaction associated with the calling thread. The timeout affects only those transactions begun with [begin\(\)](#) after the timeout is set. The `seconds` parameter sets the number of seconds from the time the transaction is begun that it waits for completion before being rolled back; if the `seconds` parameter is zero, no timeout is set for the transaction.

## Parameters

`seconds` The number of seconds that the transaction waits for completion before rolling back.

## See Also

[CosTransactions::Current](#)  
[CosTransactions::Current::begin\(\)](#)  
[CosTransactions::Current::get\\_timeout\(\)](#)

## Current::suspend()

```
// Java
org.omg.CosTransactions.Control suspend();
```

`suspend()` suspends the transaction associated with the calling thread. An identifier for the suspended transaction is returned by the method. This identifier can be passed to [resume\(\)](#) to resume the suspended transaction.

## See Also

[CosTransactions::Current](#)  
[CosTransactions::Current::resume\(\)](#)



# CosTransactions:: RecoveryCoordinator Class

The `RecoveryCoordinator` class enables a recoverable object to control the recovery process for an associated resource. A `RecoveryCoordinator` object can be obtained for a recoverable object via the [Coordinator](#) object associated with the recoverable object. [Coordinator::register\\_resource\(\)](#) returns a `RecoveryCoordinator` object.

```
// Java
package org.omg.CosTransactions;

public interface RecoveryCoordinator
    extends RecoveryCoordinatorOperations,
           org.omg.CORBA.Object,
           org.omg.CORBA.portable.IDLEntity
{
}
```

The `RecoveryCoordinator` class extends `RecoveryCoordinatorOperations`:

```
public interface RecoveryCoordinatorOperations
{
    org.omg.CosTransactions.Status replay_completion(
        org.omg.CosTransactions.Resource r
    ) throws org.omg.CosTransactions.NotPrepared;
}
```

See Also

[CosTransactions::Resource](#)

## RecoveryCoordinator::replay\_completion()

```
// Java
org.omg.CosTransactions.Status replay_completion(
    org.omg.CosTransactions.Resource r
) throws org.omg.CosTransactions.NotPrepared;
```

---

`replay_completion()` notifies the recovery coordinator that the `commit()` or `rollback()` operations have not been performed for the associated resource. Notifying the coordinator that the resource has not completed causes completion to be retried, which is useful in certain failure cases. The method returns the current status of the transaction.

### Parameters

`resource`            The resource associated with the recovery coordinator.

### Exceptions

[NotPrepared](#)        The resource is not in the prepared state.

### See Also

[CosTransactions::Resource](#)  
[CosTransactions::Status](#)

# CosTransactions::Resource Class

The `Resource` class represents a recoverable resource, that is, a transaction participant that manages data subject to change within a transaction. The `Resource` class specifies the protocol that must be defined for a recoverable resource. Interfaces that inherit from this class must implement each of the member methods to manage the data appropriately for the recoverable object based on the outcome of the transaction. These methods are invoked by the Transaction Service to execute two-phase commit; the requirements of these methods are described in the following sections.

To become a participant in a transaction, a `Resource` object must be registered with that transaction. [Coordinator::register\\_resource\(\)](#) can be used to register a resource for the transaction associated with the [Coordinator](#) object.

The full name for the class is `CosTransactions::Resource`.

```
// Java
package org.omg.CosTransactions;

public interface Resource
    extends ResourceOperations,
           org.omg.CORBA.Object,
           org.omg.CORBA.portable.IDLEntity
{
}
```

The `Resource` class extends `ResourceOperations`:

```
public interface ResourceOperations
{
    org.omg.CosTransactions.Vote prepare() throws
    org.omg.CosTransactions.HeuristicMixed,
    org.omg.CosTransactions.HeuristicHazard;

    void rollback() throws
    org.omg.CosTransactions.HeuristicCommit,
    org.omg.CosTransactions.HeuristicMixed,
    org.omg.CosTransactions.HeuristicHazard;
}
```

---

```
    void commit() throws org.omg.CosTransactions.NotPrepared,  
    org.omg.CosTransactions.HeuristicRollback,  
    org.omg.CosTransactions.HeuristicMixed,  
    org.omg.CosTransactions.HeuristicHazard;  
  
    void commit_one_phase() throws  
    org.omg.CosTransactions.HeuristicHazard;  
  
    void forget();  
  
}
```

## See Also

[CosTransactions::Synchronization](#)  
[CosTransactions::RecoveryCoordinator](#)  
[CosTransactions::Vote](#)

## Two-phase Commit

The two-phase commit requires methods `prepare()` and `commit()`.

`prepare()` must be defined to vote on the outcome of the transaction with which the resource is registered. The transaction service invokes this method as the first phase of a two-phase commit; the return value controls the second phase:

- Returns `VoteReadOnly` if the resource's data is not modified by the transaction. The transaction service does not invoke any other methods on the resource, and the resource can forget all knowledge of the transaction.
- Returns `VoteCommit` if the resource's data is written to stable storage by the transaction and the transaction is prepared. Based on the outcome of other participants in the transaction, the transaction service calls either `commit()` or `rollback()` for the resource. The resource should store a reference to the [RecoveryCoordinator](#) object in stable storage to support recovery of the resource.
- Returns `VoteRollback` for all other situations. The transaction service calls `rollback()` for the resource, and the resource can forget all knowledge of the transaction.

---

`commit()` must be defined to commit all changes made to the resource as part of the transaction. If `forget()` has already been called, no changes need to be committed. If the resource has not been prepared, the [NotPrepared](#) exception must be thrown.

Use the heuristic outcome exceptions to report heuristic decisions related to the resource. The resource must remember heuristic outcomes until `forget()` is called, so that the same outcome can be returned if the transaction service calls `commit()` again.

## One-phase Commit

`commit_one_phase()` must be defined to commit all changes made to the resource as part of the transaction. The transaction service may invoke this method if the resource is the only participant in the transaction. Unlike `commit()`, `commit_one_phase()` does not require that the resource be prepared first. Use the heuristic outcome exceptions to report heuristic decisions related to the resource. The resource must remember heuristic outcomes until `forget()` is called, so that the same outcome can be returned if the transaction service calls `commit_one_phase()` again.

## Rollback Transaction

`rollback()` must be defined to undo all changes made to the resource as part of the transaction. If `forget()` has been called, no changes need to be undone. Use the heuristic outcome exceptions to report heuristic decisions related to the resource. The resource must remember heuristic outcomes until `forget()` is called, so that the same outcome can be returned if the transaction service calls `rollback()` again.

## Forget Transaction

`forget()` must be defined to cause the resource to forget all knowledge of the transaction. The transaction service invokes this method if the resource throws a heuristic outcome exception in response to `commit()` or `rollback()`.



# CosTransactions:: SubtransactionAwareResource Class

---

**Note:** This class is not supported in this release of OTS for Orbix. The information in this section therefore does not apply to this release.

---

The `SubtransactionAwareResource` class represents a recoverable resource that makes use of nested transactions. This specialized resource object allows the resource to be notified when a subtransaction for which it is registered either commits or rolls back.

The `SubtransactionAwareResource` class specifies the protocol that must be defined for this type of recoverable resource. Interfaces that inherit from this class must implement each of the member methods to manage the recoverable object's data appropriately based on the outcome of the subtransaction. These methods are invoked by the transaction service; the requirements of these methods are described below.

[Coordinator::register\\_subtran\\_aware\(\)](#) can be used to register a resource with the subtransaction associated with the [Coordinator](#) object. The resource can also register with the top-level transaction by using [Coordinator::register\\_resource\(\)](#) as well. In this case, the protocol for the [Resource](#) class must be defined in addition to the protocol for `SubtransactionAwareResource`. See the reference page for the [Resource](#) class for more information.

```
// Java
package org.omg.CosTransactions;

public interface SubtransactionAwareResource
    extends SubtransactionAwareResourceOperations,
           org.omg.CosTransactions.Resource
{
}
```

---

The `SubtransactionAwareResource` class extends `SubtransactionAwareResourceOperations`:

```
package org.omg.CosTransactions;

public interface SubtransactionAwareResourceOperations
extends
    org.omg.CosTransactions.ResourceOperations
{
    void commit_subtransaction(
        org.omg.CosTransactions.Coordinator parent
    );

    void rollback_subtransaction();
}
```

**See Also**

[CosTransactions::Coordinator](#)  
[CosTransactions::Resource](#)  
[CosTransactions::Status](#)

## Commit Subtransaction

`commit_subtransaction()` must be defined to commit all changes made to the resource as part of the subtransaction. If an ancestor transaction rolls back, the subtransaction's changes are rolled back. The transaction service invokes this method if the resource is registered with a subtransaction and it is committed.

The method must be defined to take a [Coordinator](#) object as its only argument. When the transaction service invokes this method, it passes the [Coordinator](#) object associated with the parent transaction.

## Rollback Subtransaction

`rollback_subtransaction()` must be defined to undo all changes made to the resource as part of the subtransaction. The transaction service invokes this method if the resource is registered with a subtransaction and it is rolled back.



# CosTransactions::Synchronization Class

The `Synchronization` class represents a non-recoverable object that maintains transient state data and is dependent on a recoverable object to ensure that the data is persistent. To make data persistent, a synchronization object moves its data to one or more resources before the transaction completes.

The `Synchronization` class specifies a protocol that must be defined for this type of object. A synchronization object must be implemented as a class derived from the `Synchronization` class. The derived class must implement each of the member methods to ensure that the data maintained by the nonrecoverable object is made recoverable. The transaction service invokes these methods before and after the registered resources commit; the specific requirements of these methods are described in the following sections.

[`Coordinator::register\_synchronization\(\)`](#) can be used to register a synchronization object with the transaction associated with the [`Coordinator`](#) object.

```
// Java
package org.omg.CosTransactions;

public interface Synchronization
    extends SynchronizationOperations,
           org.omg.CosTransactions.TransactionalObject
{
}
```

The `Synchronization` class extends `SynchronizationOperations`:

```
public interface SynchronizationOperations
extends
    org.omg.CosTransactions.TransactionalObjectOperations
{
    void before_completion();

    void after_completion(
```

---

```
        org.omg.CosTransactions.Status s
    );
}
```

## Before Completion

`before_completion()` must be defined to move the synchronization object's data to a recoverable object. The transaction service invokes this method prior to the prepare phase of the transaction. The method is invoked only if the synchronization object is registered with a transaction and the transaction attempts to commit.

The only exceptions this method can throw are `CORBA::SystemException` exceptions. Throwing other exceptions can cause the transaction to be marked for rollback only.

## After Completion

`after_completion()` must be defined to do any necessary processing required by the synchronization object; for example, the method could be used to release locks held by the transaction. The transaction service invokes this method after the outcome of the transaction is complete. The method is invoked only if the synchronization object is registered with a transaction and the transaction has either committed or rolled back.

The method must be defined to take a [Status](#) value as its only argument. When the transaction service invokes this method, it passes the status of the transaction with which the synchronization object is registered.

The only exceptions this method can throw are `CORBA::SystemException` exceptions. Any exceptions that are thrown have no effect on the commitment of the transaction.

### See Also

[CosTransactions::Coordinator](#)  
[CosTransactions::Coordinator::register\\_synchronization\(\)](#)  
[CosTransactions::Resource](#)  
[CosTransactions::Status](#)

# CosTransactions::Terminator Class

The `Terminator` class enables explicit termination of a factory-created transaction. The transaction with which the `Terminator` object is associated can be either committed or rolled back. [Control::get\\_terminator\(\)](#) can be used to return the `Terminator` object associated with a transaction// Java  
package org.omg.CosTransactions;

```
public interface Terminator
    extends TerminatorOperations,
        org.omg.CORBA.Object,
        org.omg.CORBA.portable.IDLEntity
{
}
```

The `Terminator` class extends `TerminatorOperations`:

```
public interface TerminatorOperations
{
    void commit(
        boolean report_heuristics
    ) throws org.omg.CosTransactions.HeuristicMixed,
        org.omg.CosTransactions.HeuristicHazard;

    void rollback();
}
```

## See Also

[CosTransactions::Coordinator](#)  
[CosTransactions::Control::get\\_terminator\(\)](#)  
[CosTransactions::Control](#)  
[CosTransactions::Status](#)

## Terminator::commit()

```
// Java
void commit(
```

---

```
        boolean report_heuristics
    ) throws org.omg.CosTransactions.HeuristicMixed,
        org.omg.CosTransactions.HeuristicHazard;
```

`commit()` attempts to commit the transaction associated with the `Terminator` object. If the `report_heuristics` parameter is true, the [HeuristicHazard](#) exception is thrown when the participants report that a heuristic decision has possibly been made.

### Parameters

`report_heuristics` Specifies whether to report heuristic decisions for the commit.

### Exceptions

`HeuristicMixed` The transaction has been marked as rollback-only, or all participants in the transaction do not agree to commit.

### See Also

[CosTransactions::Coordinator](#)  
[CosTransactions::Terminator](#)  
[CosTransactions::Terminator::rollback\(\)](#)  
[CosTransactions::Control](#)

## Terminator::rollback()

```
// Java
void rollback();
```

`rollback()` rolls back the transaction associated with the `Terminator` object.

### See Also

[CosTransactions::Coordinator](#)  
[CosTransactions::Terminator](#)  
[CosTransactions::Terminator::commit\(\)](#)

# CosTransactions::TransactionalObject Class

The TransactionalObject interface has been deprecated and replaced with transactional policies (see [“OTSPolicyValue Data Type” on page 570](#)). Backward compatibility with existing OTS implementations is provided for outbound requests only and only if the target object does not have a transactional policy in its IOR.

See the *CORBA Programmer's Guide* for details of interoperability with existing OTS implementations.

```
// Java
package org.omg.CosTransactions;

public interface TransactionalObject
    extends TransactionalObjectOperations,
           org.omg.CORBA.Object,
           org.omg.CORBA.portable.IDLEntity
{
}
```

The TransactionalObject class extends TransactionalObjectOperations:

```
public interface TransactionalObjectOperations
{
}
```



# CosTransactions::TransactionFactory Class

The `TransactionFactory` class represents a transaction factory that allows the originator of transactions to begin a new transaction for use with the explicit model of transaction demarcation. Servers provide a default instance of this class. Clients can bind to the default instance by using the standard binding mechanism for the object request broker.

```
// Java
package org.omg.CosTransactions;

public interface TransactionFactory
    extends TransactionFactoryOperations,
           org.omg.CORBA.Object,
           org.omg.CORBA.portable.IDLEntity
{
}
```

The `TransactionFactory` class extends `TransactionFactoryOperations`:

```
public interface TransactionFactoryOperations
{
    org.omg.CosTransactions.Control create(
        int time_out
    );

    org.omg.CosTransactions.Control recreate(
        org.omg.CosTransactions.PropagationContext ctx
    );
}
```

**See Also**

[CosTransactions::Control](#)

---

## TransactionFactory::create()

```
// Java
org.omg.CosTransactions.Control create(
    int time_out
);
```

`create()` creates a new top-level transaction for use with the explicit model of transaction demarcation. A [Control](#) object is returned for the transaction. The [Control](#) object can be used to propagate the transaction context. See the reference page for the [Control](#) class for more information.

### Parameters

<code>timeout</code>	Specifies the number of seconds that the transaction waits to complete before rolling back. If the <code>timeout</code> parameter is zero, no timeout is set for the transaction.
----------------------	---

### See Also

[CosTransactions::TransactionFactory](#)  
[CosTransactions::Control](#)

## TransactionFactory::recreate()

```
// Java
org.omg.CosTransactions.Control recreate(
    org.omg.CosTransactions.PropagationContext ctx
);
```

Creates a new representation for an existing transaction defined in the propagation context `ctx`. This is used to import a transaction from another domain. The method returns a control object for the new transaction representation.

### See Also

[CosTransactions::Coordinator::get\\_txcontext\(\)](#)



# CosTypedEventChannelAdmin Module

The `CosTypedEventChannelAdmin` module defines the interfaces for making connections between suppliers and consumers that use either generic or typed communication. Its interfaces are specializations of the corresponding interfaces in the `CosEventChannel` module.

---

**Note:** IONA's implementation of typed events only supports the typed push style of event communication. The `TypedProxyPullSupplier` interface, the `TypedSupplierAdmin::obtain_typed_pull_consumer()` operation, and the `TypedConsumerAdmin::obtain_typed_pull_supplier()` operation are **not** implemented.

---

## CosTypedEventChannelAdmin Exceptions

### `CosTypedEventChannelAdmin::InterfaceNotSupported`

```
exception InterfaceNotSupported {};
```

`InterfaceNotSupported` is raised when an attempt to obtain a `TypedProxyPushConsumer` fails to find an implementation that supports the strongly typed interface required by the client.

### `CosTypedEventChannelAdmin::NoSuchImplementation`

```
exception NoSuchImplementation {};
```

`NoSuchImplementation` is raised when an attempt to obtain a `ProxyPushSupplier` fails to find an implementation that supports the strongly typed interface required by the client.

---

## CostypedEventChannelAdmin Data Types

### **CostypedEventChannelAdmin::Key Type**

```
typedef string Key;
```

A string that holds the interface repository ID of the strongly typed interface used by a typed event client.

# CosTypedEventChannelAdmin:: TypedConsumerAdmin Interface

```
interface TypedConsumerAdmin : CosEventChannelAdmin::ConsumerAdmin
{
    TypedProxyPullSupplier obtain_typed_pull_supplier(
        in Key supported_interface)
    raises (InterfaceNotSupported);

    CosEventChannelAdmin::ProxyPushSupplier
    obtain_typed_push_supplier(in Key uses_interface)
    raises (NoSuchImplementation);
};
```

The `TypedConsumerAdmin` interface extends the functionality of the generic `ConsumerAdmin` to support connecting consumer to a typed event channel.

## **TypedConsumerAdmin::obtain\_typed\_pull\_supplier()**

```
TypedProxyPullSupplier obtain_typed_pull_supplier(
    in Key supported_interface)
raises (InterfaceNotSupported);
```

The `obtain_typed_pull_supplier()` operation returns a `TypedProxyPullSupplier` that supports the interface `Pull<supported_interface>`.

### **Parameters**

`supported_interface` Specifies the interface which the returned `TypedProxyPullSupplier` must support.

### **Exceptions**

`InterfaceNotSupported` Raised if `TypedProxyPullSupplier` implementation supporting the specified interface is available.

---

## TypedConsumerAdmin::obtain\_typed\_push\_supplier()

```
CosEventChannelAdmin::ProxyPushSupplier  
    obtain_typed_push_supplier(in Key uses_interface)  
    raises (NoSuchImplementation);
```

The `obtain_typed_push_supplier()` operation returns a `ProxyPushSupplier` that makes calls on interface `uses_interface`.

### Parameters

`uses_interface` Specifies the interface on which the returned `ProxyPushSupplier` must make calls.

### Exceptions

`NoSuchImplementation` Raised if no `ProxyPushConsumer` can be found that supports the specified interface.

## Unsupported Operations

The Application Server Platform does not support the typed pull model or the connection of generic consumers to a typed event channel. Therefore, a `TypedConsumerAdmin` object will throw `NO_IMPLEMENT` for the following operations:

- `obtain_typed_pull_supplier()`
- `obtain_push_supplier()`
- `obtain_pull_supplier()`

# CosTypedEventChannelAdmin:: TypedEventChannel Interface

```
interface TypedEventChannel
{
    TypedConsumerAdmin for_consumers();

    TypedSupplierAdmin for_suppliers();

    void destroy();
};
```

This interface is the equivalent of `CosEventChannelAdmin::EventChannel` for typed events. It provides a factory for `TypedConsumerAdmin` objects and `TypedSupplierAdmin` objects. Both of which are capable of providing proxies for typed communication.



# CosTypedEventChannelAdmin:: TypedProxyPushConsumer Interface

```
interface TypedProxyPushConsumer :  
    CosEventChannelAdmin::ProxyPushConsumer,  
    CosTypedEventComm::TypedPushConsumer  
{  
};
```

The `TypedProxyPushConsumer` interface extends the functionality of the `ProxyPushConsumer` to support connecting push suppliers to a typed event channel.

By inheriting from `CosEventChannelAdmin::ProxyPushConsumer`, this interface supports:

- connection and disconnection of push suppliers.
- generic push operation.

By inheriting from `CosTypedEventComm::TypedPushConsumer`, it extends the functionality of the generic `ProxyPushConsumer` to enable its associated supplier to use typed push communication. When a reference to a `TypedProxyPushConsumer` is returned by `get_typed_consumer()`, it has the interface identified by the `key`.

## Unsupported Operations

The `TypedProxyPushConsumer` reference will throw `NO_IMPLEMENT` for the `push()` operation. A supplier should instead call `push()` on the reference it obtains from the `get_typed_consumer()` operation.





# CosTypedEventChannelAdmin:: TypedSupplierAdmin Interface

```
interface TypedSupplierAdmin : CosEventChannelAdmin::SupplierAdmin
{
    TypedProxyPushConsumer obtain_typed_push_consumer(
        in Key supported_interface)
        raises (InterfaceNotSupported);

    CosEventChannelAdmin::ProxyPullConsumer
        obtain_typed_pull_consumer(in Key uses_interface)
        raises (NoSuchImplementation);
};
```

The `TypedSupplierAdmin` interface extends the functionality of the generic `SupplierAdmin` to support connecting suppliers to a typed event channel.

## **TypedSupplierAdmin::obtain\_typed\_push\_consumer()**

```
TypedProxyPushConsumer obtain_typed_push_consumer(
    in Key supported_interface)
    raises (InterfaceNotSupported);
```

The `obtain_typed_push_consumer()` operation returns a `TypedProxyPushConsumer` that supports the specified interface.

### **Parameters**

`supported_interface` Specifies the interface that the returned `TypedProxyPushConsumer` must support.

### **Exceptions**

`InterfaceNotSupported` Raised if no consumer implementation supporting the specified interface is available.

---

## TypedSupplierAdmin::obtain\_typed\_pull\_consumer()

```
CosEventChannelAdmin::ProxyPullConsumer  
    obtain_typed_pull_consumer(in Key uses_interface)  
    raises (NoSuchImplementation);
```

The `obtain_typed_pull_consumer()` operation returns a `ProxyPullConsumer` that calls operations in the interface `Pull<uses_interface>`.

### Parameters

`uses_interface` Specifies the interface which the returned `ProxyPullConsumer` must support.

### Exceptions

`NoSuchImplementation` Raised if no `ProxyPullConsumer` can be found that supports the specified interface.

## Unsupported Operations

The Application Server Platform does not support the typed pull model or the connection of generic suppliers to a typed event channel. Therefore, the `TypedSupplierAdmin` reference will throw `NO_IMPLEMENT` for the following operations:

- `obtain_typed_pull_consumer()`
- `obtain_push_consumer()`
- `obtain_pull_consumer()`

# CosTypedEventComm Module

This module specifies two interfaces used to support typed event communication. `TypedPushConsumer` supports push style typed event communication. Typed event clients retain the capability to use generic event communication.

---

**Note:** IONA's implementation of typed events only supports typed push style events. The `TypedPullSupplier` interface is **not** implemented.

---



# CosTypedEventComm:: TypedPushConsumer Interface

```
interface TypedPushConsumer : CosEventComm::PushConsumer
{
    Object get_typed_consumer();
};
```

The `TypedPushConsumer` interface is used to implement push-style consumers that wish to participate in typed event communication. By inheriting from the generic `PushConsumer` interface, this interface retains the ability to participate in generic push-style event communication. This inheritance also requires that `TypedPushConsumer` objects implement the generic `push()` operation. However, if the consumer will be used solely for typed event communication, the `push()` implementation can simply raise the standard CORBA exception `NO_IMPLEMENT`.

## **TypedPushConsumer::get\_typed\_consumer()**

```
Object get_typed_consumer();
```

`get_typed_consumer()` returns a reference to a typed push consumer. This reference is returned as a reference to type `Object` and must be narrowed to the appropriate interface. If the push supplier and the typed push consumer do not support the same interface, the `narrow()` will fail.



# CSI Overview

The CSI module defines the basic data types needed for the OMG Common Secure Interoperability (CSIv2) specification. This reference page is a partial extract from the CSI module that includes only the data types needed for the IT\_CSI module.

## CSI::OID Sequence

```
typedef sequence <octet> OID;  
    // ASN.1 Encoding of an OBJECT IDENTIFIER
```

The type that represents an ASN.1 object identifier in binary format.

## CSI::OIDList Sequence

```
typedef sequence <OID> OIDList;
```

The type that represents a list of ASN.1 object identifiers.

## CSI::GSS\_NT\_ExportedName

```
typedef sequence <octet> GSS_NT_ExportedName;
```

An encoding of a GSS Mechanism-Independent Exported Name Object as defined in [IETF RFC 2743] Section 3.2, "GSS Mechanism-Independent Exported Name Object Format," p. 84. See <http://www.ietf.org/rfc/rfc2743.txt>.

### See Also

```
IT_CSI::AuthenticationServicePolicy::target_name
```

## CSI::IdentityTokenType

```
typedef unsigned long IdentityTokenType;
```

---

The type of a CSlv2 identity token.

**See Also**

`CSI::IdentityToken`

### **CSI::ITTAbsent**

```
const IdentityTokenType ITTAbsent = 0;
```

The identity token is absent. This indicates that the invocation is not being made on behalf of another principal.

**See Also**

`CSI::IdentityToken`

### **CSI::ITTAnonymous**

```
const IdentityTokenType ITTAnonymous = 1;
```

Indicates that the invocation is being made on behalf of an unidentified and unauthenticated principal.

**See Also**

`CSI::IdentityToken`

### **CSI::ITTPrincipalName**

```
const IdentityTokenType ITTPrincipalName = 2;
```

Indicates that the invocation is being made on behalf of an identifiable and authenticated principal.

**See Also**

`CSI::IdentityToken`

### **CSI::ITTX509CertChain**

```
const IdentityTokenType ITTX509CertChain = 4;
```

Not used in the current implementation of CSlv2.

**See Also**

`CSI::IdentityToken`



---

## CSI::ITTDistinguishedName

```
const IdentityTokenType ITTDistinguishedName = 8;
```

Not used in the current implementation of CSIPv2.

### See Also

CSI::IdentityToken

## CSI::IdentityExtension

```
typedef sequence <octet> IdentityExtension;
```

A data type that enables the range of identity tokens to be extended. The OMG reserves this type for future extensions.

### See Also

CSI::IdentityToken

## CSI::IdentityToken Union

```
union IdentityToken switch ( IdentityTokenType ) {  
    case ITTAbsent: boolean absent;  
    case ITTAnonymous: boolean anonymous;  
    case ITTPrincipalName: GSS_NT_ExportedName principal_name;  
    case ITTX509CertChain: X509CertificateChain certificate_chain;  
    case ITTDistinguishedName: X501DistinguishedName dn;  
    default: IdentityExtension id;  
};
```

The type that is used to represent an identity token. Only the following identity token types are currently used by Orbix:

- ITTAbsent
- ITTAnonymous
- ITTPrincipalName

---

## CSI::StringOID

```
typedef string StringOID;
```

This type is the string representation of an ASN.1 OBJECT IDENTIFIER (OID). OIDs are represented by the string `oid:` followed by the integer base-10 representation of the OID separated by dots. For example, the OID corresponding to the OMG is represented as: `"oid:2.23.130"`

## CSI::GSS\_NT\_Export\_Name\_OID

```
const StringOID GSS_NT_Export_Name_OID = "oid:1.3.6.1.5.6.4";
```

The GSS Object Identifier for name objects of the Mechanism-Independent Exported Name Object type is:

```
{ iso(1) org(3) dod(6) internet(1) security(5) nametypes(6)
  gss-api-exported-name(4) }
```

# CSIIOP Overview

The CSI inter-ORB protocol (CSIIOP) IDL module defines the data types that are used for encoding the CSv2 service contexts and IOR components . This reference page is a partial extract from the CSIIOP module that includes only the data types needed for the `IT_CSI` module.

## **CSIIOP::AssociationOptions**

```
typedef unsigned short AssociationOptions;
```

The type used to define association option flags.

## **CSIIOP::NoProtection**

```
const AssociationOptions NoProtection = 1;
```

Not needed in the current implementation of CSv2.

## **CSIIOP::Integrity**

```
const AssociationOptions Integrity = 2;
```

Not needed in the current implementation of CSv2.

## **CSIIOP::Confidentiality**

```
const AssociationOptions Confidentiality = 4;
```

Not needed in the current implementation of CSv2.

---

## **CSIIOP::DetectReplay**

```
const AssociationOptions DetectReplay = 8;
```

Not needed in the current implementation of CSv2.

## **CSIIOP::DetectMisordering**

```
const AssociationOptions DetectMisordering = 16;
```

Not needed in the current implementation of CSv2.

## **CSIIOP::EstablishTrustInTarget**

```
const AssociationOptions EstablishTrustInTarget = 32;
```

Not needed in the current implementation of CSv2.

## **CSIIOP::EstablishTrustInClient**

```
const AssociationOptions EstablishTrustInClient = 64;
```

The `EstablishTrustInClient` association option can be specified in the `support` attribute or in the `target_requires` attribute of the `IT_CSI::AuthenticationServicePolicy` policy. This policy enables you to specify that a client or server can require and support client authentication over the transport using CSv2.

### **See Also**

```
IT_CSI::AuthenticationService  
IT_CSI::AuthenticationServicePolicy
```

## **CSIIOP::NoDelegation**

```
const AssociationOptions NoDelegation = 128;
```

Not supported in the current implementation of CSv2.

---

## CSIIOP::SimpleDelegation

```
const AssociationOptions SimpleDelegation = 256;
```

Not supported in the current implementation of CSIV2.

## CSIIOP::CompositeDelegation

```
const AssociationOptions CompositeDelegation = 512;
```

Not supported in the current implementation of CSIV2.

## CSIIOP::IdentityAssertion

```
const AssociationOptions IdentityAssertion = 1024;
```

The `IdentityAssertion` association option can be specified in the `support` attribute of the `IT_CSI::AttributeServicePolicy` policy. This policy enables you to specify that a client or server supports identity assertion (principal propagation) using CSIV2.

### See Also

```
IT_CSI::AttributeService  
IT_CSI::AttributeServicePolicy
```

## CSIIOP::DelegationByClient

```
const AssociationOptions DelegationByClient = 2048;
```

Not supported in the current implementation of CSIV2.

## CSIIOP::ServiceConfigurationSyntax Type

```
typedef unsigned long ServiceConfigurationSyntax;
```

The type used to identify a syntax for specifying privilege authority names.

The high order 20-bits of each `ServiceConfigurationSyntax` constant shall contain the Vendor Minor Codeset ID (VMCID) of the organization that defined the syntax. The low order 12 bits shall contain the

---

organization-scoped syntax identifier. The high-order 20 bits of all syntaxes defined by the OMG shall contain the VMCID allocated to the OMG (that is, 0x4F4D0).

**See Also**

`CSIIOP::ServiceConfiguration`

### **CSIIOP::SCS\_GeneralNames**

```
const ServiceConfigurationSyntax SCS_GeneralNames = CSI::OMGVMCID
    | 0;
```

Identifies the `GeneralNames` syntax (as defined in [IETF RFC 2459]) for specifying privilege authority names.

### **CSIIOP::SCS\_GSSExportedName**

```
const ServiceConfigurationSyntax SCS_GSSExportedName =
    CSI::OMGVMCID | 1;
```

Identifies the GSS exported name syntax (as defined in [IETF RFC 2743] Section 3.2) for specifying privilege authority names.

### **CSIIOP::ServiceSpecificName**

```
typedef sequence <octet> ServiceSpecificName;
```

A type that contains a privilege authority name, encoded using either the `CSIIOP::SCS_GeneralNames` or the `CSIIOP::SCS_GSSExportedName` syntax.

**See Also**

`CSIIOP::ServiceConfiguration`

### **CSIIOP::ServiceConfiguration Structure**

```
struct ServiceConfiguration {
    ServiceConfigurationSyntax syntax;
    ServiceSpecificName name;
};
```

Not used in the current implementation of CSv2.

---

## **CSIIOP::ServiceConfigurationList Sequence**

```
typedef sequence <ServiceConfiguration> ServiceConfigurationList;
```

A list of `ServiceConfiguration` structures.

Not used in the current implementation of CSv2.





# DsEventLogAdmin Module

The `DsEventLogAdmin` module defines the [EventLog](#) interface which provides logging capabilities for event service clients. This module also defines the `EventLogFactory` interface which is used to instantiate [EventLog](#) objects.



# DsEventLogAdmin::EventLog Interface

```
interface EventLog : DsLogAdmin::Log,  
                  CosEventChannelAdmin::EventChannel  
{  
};
```

The `EventLog` interface extends the functionality of the [Log](#) interface by also inheriting from `CosEventChannelAdmin::EventChannel`. This inheritance provides `EventLog` objects the ability to log events as they are passed through an event channel. The `EventLog` interface does not define any operations.



# DsEventLogAdmin::EventLogFactory Interface

The `EventLogFactory` interface defines two operations for instantiating [EventLog](#) objects.

## `EventLogFactory::create()`

```
EventLog create(in LogFullActionType full_action,  
               in unsigned long long max_size,  
               in DsLogAdmin::CapacityAlarmThresholdList thresholds,  
               out LogId id);  
raises (InvalidLogFullAction  
       InvalidThreshold);
```

Returns an instantiated [EventLog](#) object. The [LogId](#) returned is assigned by the service and can be used to access the returned [EventLog](#) object.

### Parameters

<code>full_action</code>	Specifies what the log object will do when it fills up.
<code>max_size</code>	Specifies the maximum amount of data, in bytes, the log can hold.
<code>thresholds</code>	Specifies , as a percentage of max log size, the points at which an <a href="#">ThresholdAlarm</a> event will be generated.
<code>id</code>	The <a href="#">LogId</a> assigned to the <code>EventLog</code> object by the service.

### Exceptions

[InvalidLogFullAction](#) The specified `full_action` is not a valid [LogFullActionType](#).

[InvalidThreshold](#) One of the thresholds specified is invalid.

---

## EventLogFactory::create\_with\_id()

```
EventLog create_with_id(in LogId id,  
                        in LogFullActionType full_action,  
                        in unsigned long long max_size)  
                        in DsLogAdmin::CapacityAlarmThresholdList thresholds)  
raises(DsLogAdmin::LogIdAlreadyExists,  
       DsLogAdmin::InvalidLogFullAction,  
       DsLogAdmin::InvalidThreshold);
```

Returns an instantiated [EventLog](#) object with a user supplied id.

### Parameters

id	Specifies the <a href="#">LogId</a> to assign the <a href="#">EventLog</a> .
full_action	Specifies what the log object will do when it fills up.
max_size	Specifies the maximum amount of data, in bytes, the log can hold.
thresholds	Specifies , as a percentage of max log size, the points at which an <a href="#">ThresholdAlarm</a> event will be generated.

### Exceptions

<a href="#">LogIdAlreadyExists</a>	A log with the specified id already exists.
<a href="#">InvalidLogFullAction</a>	The specified full_action is not a valid <a href="#">LogFullActionType</a> .
<a href="#">InvalidThreshold</a>	One of the thresholds specified is invalid.

# DsLogAdmin Module

DsLogAdmin specifies the `Log` interfaces which forms the basis for the `BasicLog` interface, `EventLog` interface, and the `NotifyLog` interface. DsLogAdmin also specifies the [BasicLog](#) and [BasicLogFactory](#) to support the basic logging service. In addition, this module specifies the `Iterator` interface to support the iterators returned when retrieving records from a log.

This module also specifies all of the exceptions and major datatypes used by the telecom logging service.

## DsLogAdmin Exceptions

### DsLogAdmin::InvalidParam Exception

```
exception InvalidParam {string details};
```

Raised when an illegal value is used to set a log's properties. It contains the name of the property being set and the illegal value.

### DsLogAdmin::InvalidThreshold Exception

```
exception InvalidThreshold {};
```

Raised when an attempt is made to set a threshold alarm at a value outside the range of 0%-99%.

### DsLogAdmin::InvalidTime Exception

```
exception InvalidTime{};
```

Raised by `set_week_mask()` when one of the values specified for a start or stop time is not within the valid range.

---

## DsLogAdmin::InvalidTimeInterval Exception

```
exception InvalidTimeInterval{};
```

Raised by `set_week_mask()` when one of the time intervals used to set a log's schedule is improperly formed. For example, the stop time is before the start. Also raised if the intervals overlap.

## DsLogAdmin::InvalidMask Exception

```
exception InvalidMask{};
```

Raised by `set_week_mask()` when the days parameter used in setting a log's schedule is malformed.

## DsLogAdmin::LogIdAlreadyExists Exception

```
exception LogIdAlreadyExists{};
```

Raised by `create_with_id()` if an attempt is made to create a log with an id that is already in use.

## DsLogAdmin::InvalidGrammar Exception

```
exception InvalidGrammar{};
```

Raised by `query()` and `delete_records()` if an unsupported constraint grammar is specified. The grammar implemented in Iona's telecom logging service is `EXTENDED_TCL`.

## DsLogAdmin::InvalidConstraint Exception

```
exception InvalidConstraint{};
```

Raised by `query()` and `delete_records()` if a constraint expression is not syntactically correct according to the specified grammar.



### **DsLogAdmin::LogFull Exception**

```
exception LogFull{short n_records_written};
```

Raised when an attempt is made to log records in a log that is full and has its `full_action` set to `halt`. It returns the number of records that were successfully written to the log.

### **DsLogAdmin::LogOffDuty Exception**

```
exception LogOffDuty{};
```

Raised when an attempt is made to log records in a log whose availability status is off duty.

### **DsLogAdmin::LogLocked Exception**

```
exception LogLocked{};
```

Raised when an attempt is made to log records in a log whose administrative state is `locked`.

### **DsLogAdmin::LogDisabled Exception**

```
exception LogDisabled{};
```

Raised when an attempt is made to log records in a log whose operational state is `disabled`.

### **DsLogAdmin::InvalidRecordId Exception**

```
exception InvalidRecordId{};
```

Raised when the record id specified does not exist in the log.

---

## DsLogAdmin::InvalidAttribute Exception

```
exception InvalidAttribute{string attr_name; any value};
```

Raised when one of the attributes set on a record is invalid. It returns the name of the invalid attribute and the value specified for it.

## DsLogAdmin::InvalidLogFullAction Exception

```
exception InvalidLogFullAction{};
```

Raised if an attempt is made to set a log's `full_action` to a value other than `wrap` or `halt`.

## DsLogAdmin::UnsupportedQoS Exception

```
exception UnsupportedQoS{QoSList denied};
```

# DsLogAdmin Constants

`DsLogAdmin` defines the majority of the constant values used when developing a telecom logging service application.

## Querying Constants

`DsLogAdmin` defines one constant to support queries:

```
const string default_grammar = "EXTENDED_TCL";
```

## Full Action Constants

Two constants are defined to support a log's `full_action`:

```
const LogFullActionType wrap = 0;  
const LogFullActionType halt = 1;
```

## Scheduling Constants

DsLogAdmin defines the following constants to support log scheduling:

```
const unsigned short Sunday    = 1;
const unsigned short Monday    = 2;
const unsigned short Tuesday   = 4;
const unsigned short Wednesday = 8;
const unsigned short Thursday  = 16;
const unsigned short Friday    = 32;
const unsigned short Saturday  = 64;
```

## QoS Constants

DsLogAdmin defines the following constants to support log QoS properties:

```
const QoSType QoSNone = 0;
const QoSType QoSFlush = 1;
const QoSType QoSReliable = 2;
```

# DsLogAdmin Datatypes

## DsLogAdmin::LogId Type

```
typedef unsigned long LogId;
```

Specifies a log's unique id. The id is used by several methods for specifying which log to use or to locate a specific log.

## DsLogAdmin::RecordId Type

```
typedef unsigned long long RecordId;
```

Specifies a record's id. A record's id is unique within the log storing it.

---

## DsLogAdmin::RecordIdList Sequence

```
typedef sequence<RecordId> RecordIdList;
```

Specifies a list of record ids. The list does not need to be in any particular order.

## DsLogAdmin::Constraint Type

```
typedef string Constraint;
```

Specifies the constraints used for querying a log's records.

## DsLogAdmin::TimeT Type

```
typedef TimeBase::TimeT TimeT;
```

Used to record logging times and for setting a log's duration.

## DsLogAdmin::NVPair Structure

```
struct NVPair
{
    string name;
    any    value;
};
```

Specifies a name/value pair used to construct attributes for records.

name                   The name of the attribute. The value can be any string.

value                   An any containing the setting for the attribute.

## DsLogAdmin::NVList Sequence

```
typedef sequence<NVPair> NVList;
```

A list of name/value record attributes.

## DsLogAdmin::TimeInterval Structure

```
struct TimeInterval
{
    TimeT start;
    TimeT stop;
};
```

Specifies the start and stop times for a logging session.

start	The start time for the current logging session.
stop	The end time for the current logging session.

## DsLogAdmin::LogRecord Structure

```
struct LogRecord
{
    RecordId id;
    TimeT time;
    NVList attr_list;
    any info;
};
```

The data stored when a new record is logged.

id	The unique identifier for the record
time	The time at which the record was logged.
attr_list	An optional list of attributes specified by the client
info	The data contained in the record.

## DsLogAdmin::RecordList Sequence

```
typedef sequence<LogRecord> RecordList;
```

A list of records.

---

## DsLogAdmin::Anys Sequence

```
typedef sequence<any> Anys;
```

A sequence of data stored in individual `any` packages.

## DsLogAdmin::AvailabilityStatus Structure

```
struct AvailabilityStatus
{
    boolean off_duty;
    boolean log_full;
};
```

Represents the availability of a log.

<code>off_duty</code>	<code>true</code> means the log is not scheduled to accept new events. <code>false</code> means it is scheduled to receive new events.
<code>log_full</code>	If the log is full this member will be <code>true</code> .

## DsLogAdmin::LogFullActionType Type

```
typedef unsigned short LogFullActionType;
```

Specifies a log's `full_action`. It can either be `halt` or `wrap`.

## DsLogAdmin::Time24 Structure

```
struct Time24
{
    unsigned short hour; // 0-23
    unsigned short minute; // 0-59
};
```

Specifies the fine grained times for a log's schedule

<code>hour</code>	An hour specified in 24 hour format
<code>minute</code>	The minute within an hour. Can be a value from 0-59.

## DsLogAdmin::Time24Interval Structure

```
struct Time24Interval
{
    Time24 start;
    Time24 stop;
};
```

A fine grained interval during which a log is scheduled to log new records.

**start**            The time at which a log will begin logging new records.  
**stop**            The time at which a log will stop logging new records.

## DsLogAdmin::IntervalsOfDay Sequence

```
typedef sequence<Time24Interval> IntervalsOfDay;
```

A list of fine grained logging intervals.

## DsLogAdmin::DaysOfWeek Type

```
typedef unsigned short DaysOfWeek;
```

A bit mask specifying the days of the week a fine grained logging interval is valid. It is constructed using the scheduling constants listed in “Scheduling Constants” on page 645.

## DsLogAdmin::WeekMaskItem Structure

```
struct WeekMaskItem
{
    DaysOfWeek        days;
    IntervalsOfDay    intervals;
};
```

---

Specifies a fined grain log schedule.

days            A bitmask specifying the days of the week for which the specified intervals are valid.

intervals        The fine grained logging intervals.

### **DsLogAdmin::WeekMask Sequence**

```
typedef sequence<WeekMaskItem> WeekMask;
```

Specifies a log's fine grained logging schedule.

### **DsLogAdmin::Threshold Type**

```
typedef unsigned short Threshold;
```

Specifies a threshold point, in terms of a percentage of how full a log is, at which to generate an alarm. Valid values are from 0-100.

### **DsLogAdmin::CapacityAlarmThresholdList Sequence**

```
typedef sequence<Threshold> CapacityAlarmThresholdList;
```

A list of thresholds at which alarms are generated.



**DsLogAdmin::OperationalState Enum**

```
enum OperationalState {disabled, enabled};
```

Specifies if a log is ready to log new records.

**Table 10:** *Log operational states*

Operational State	Reason
enabled	The log is healthy and its full functionality is available for use.
disabled	The log has encountered a runtime error and is unavailable. The log will not accept any new records and it may not be able to retrieve valid records. The log will still attempt to forward events if its <a href="#">ForwardingState</a> is set to on.

**DsLogAdmin::AdministrativeState Enum**

```
enum AdministrativeState {locked, unlocked};
```

Specifies if a log can accept new records.

**DsLogAdmin::ForwardingState Enum**

```
enum ForwardingState {on, off}
```

Specifies if a log will forward events or not.

**DsLogAdmin::LogList Sequence**

```
typedef sequence<Log> LogList;
```

A sequence of log object references.

---

## DsLogAdmin::LogIdList Sequence

```
typedef sequence<LogId> LogIdList;
```

A sequence of log ids.

## DsLogAdmin::QoSType Type

```
typedef unsigned short QoSType;
```

Specifies the log's QoS level. Valid values are QoSNone, QoSFlush, and QoSReliable.

## DsLogAdmin::QoSList Sequence

```
typedef sequence<QoSType> QoSList;
```

A list of QoSType.

# DsLogAdmin::BasicLog Interface

The `BasicLog` interface extends the [Log](#) interface to support the logging by event-unaware CORBA objects. It defines only one method, `destroy()`, which is used to destroy a `BasicLog` object.

```
interface BasicLog : Log
{
    void destroy();
};
```



# DsLogAdmin::BasicLogFactory Interface

The BasicLogFactory interface provides the functionality to instantiate a [BasicLog](#) object.

```
interface BasicLogFactory : LogMgr
{
    BasicLog create(in LogFullActionType full_action,
                    in unsigned long long max_size,
                    out LogId id)
        raises (InvalidLogFullAction);

    BasicLog create_with_id(in LogId id,
                            in LogFullActionType full_action,
                            in unsigned long long max_size)
        raises (LogIdAlreadyExists, InvalidLogFullAction);
};
```

## BasicLogFactory::create()

```
BasicLog create(in LogFullActionType full_action,
                in unsigned long long max_size,
                out LogId id);
raises (InvalidLogFullAction);
```

Returns an instantiated [BasicLog](#) object. The [LogId](#) returned is assigned by the service and can be used to access the returned [BasicLog](#) object.

## Parameters

full_action	Specifies what the log object will do when it fills up.
max_size	Specifies the maximum amount of data, in bytes, the log can hold.
id	The LogId assigned to the BasicLog object by the service.

---

## Exceptions

[InvalidLogFullAction](#) The specified full\_action is not a valid [LogFullActionType](#).

## BasicLogFactory::create\_with\_id()

```
BasicLog create_with_id(in LogId id,  
                        in LogFullActionType full_action,  
                        in unsigned long long max_size)  
    raises (LogIdAlreadyExists, InvalidLogFullAction);
```

Returns an instantiated [BasicLog](#) object with a user supplied id.

## Parameters

id	Specifies the LogId to assign the BasicLog.
full_action	Specifies what the log object will do when it fills up.
max_size	Specifies the maximum amount of data, in bytes, the log can hold.

## Exceptions

[InvalidLogFullAction](#) The specified full\_action is not a valid [LogFullActionType](#).

[LogIdAlreadyExists](#) A log with the specified id already exists.

# DsLogAdmin::Iterator Interface

The `Iterator` interface provides the methods for accessing records returned by the iterator when querying a log. It also provides the method used to release the resources consumed by the returned iterator.

```
interface Iterator
{
    RecordList get(in unsigned long position,
                  in unsigned long how_many)
    raises(InvalidParam);

    void destroy();
};
```

## Iterator::get()

```
RecordList get(in unsigned long position,
                in unsigned long how_many)
raises(InvalidParam);
```

Retrieves the specified number of records from the iterator object and returns them as a `RecordList`.

### Parameters

<code>position</code>	The number of the record from which to start retrieving records.
<code>how_many</code>	The number of records to return.

### Exceptions

[InvalidParam](#) Raised if the position is negative or past the end of the list.

---

## **Iterator::destroy()**

```
void destroy();
```

Releases the resources used by the iterator object. If an iterator object is returned, you must explicitly destroy it.



# DsLogAdmin::Log Interface

The `Log` interface provides all of the basic functionality for log objects. All other log interfaces inherit from this interface. The `Log` interface provides the methods for managing a log's functional properties including its `full_action` and maximum size. It also defines the methods for querying the log for records, retrieving records from the log, and deleting records from the log. In addition, it defines the `flush()` method and two methods for copying logs.

```
interface Log
{
    LogMgr my\_factory\(\);
    LogId id\(\);

    unsigned long get\_max\_record\_life\(\);
    void set\_max\_record\_life(in unsigned long life);

    unsigned long long get\_max\_size\(\);
    void set\_max\_size(in unsigned long long size)
        raises (InvalidParam);
    unsigned long long get\_current\_size\(\);
    unsigned long long get\_n\_records\(\);

    LogFullActionType get\_log\_full\_action\(\);
    void set\_log\_full\_action(in LogFullActionType action)
        raises(InvalidLogFullAction);

    AdministrativeState get\_administrative\_state\(\);
    void set\_administrative\_state(in AdministrativeState state);

    ForwardingState get\_forwarding\_state\(\);
    void set\_forwarding\_state(in ForwardingState state);

    OperationalState get\_operational\_state\(\);
    AvailabilityStatus get\_availability\_status\(\);

    TimeInterval get\_interval\(\);
    void set\_interval(in TimeInterval interval)
        raises (InvalidTime, InvalidTimeInterval);
}
```

---

```

CapacityAlarmThresholdList get_capacity_alarm_thresholds();
void set_capacity_alarm_thresholds(in CapacityAlarmThresholdList
    threshs)
    raises (InvalidThreshold);

WeekMask get_week_mask();
void set_week_mask(in WeekMask masks)
    raises (InvalidTime, InvalidTimeInterval, InvalidMask);

QoSList get_log_qos();
void set_log_qos(in QoSList qos) raises (UnsupportedQoS)

RecordList query(in string grammar, in Constraint c,
    out Iterator i)
    raises(InvalidGrammar, InvalidConstraint);

RecordList retrieve(in TimeT from_time, in long how_many,
    out Iterator i);

unsigned long match(in string grammar, in Constraint c)
    raises(InvalidGrammar, InvalidConstraint);

unsigned long delete_records(in string grammar, in Constraint c)
    raises(InvalidGrammar, InvalidConstraint);
unsigned long delete_records_by_id(in RecordIdList ids);

void write_records(in Anys records)
    raises(LogFull, LogOffDuty, LogLocked, LogDisabled);
void write_recordlist(in RecordList list)
    raises(LogFull, LogOffDuty, LogLocked, LogDisabled);

void set_record_attribute(in RecordId id, in NVList attr_list)
    raises(InvalidRecordId, InvalidAttribute);
unsigned long set_records_attribute(in string grammar,
    in Constraint c,
    in NVList attr_list)
    raises(InvalidGrammar, InvalidConstraint, InvalidAttribute);

NVList get_record_attribute(in RecordId id)
    raises(InvalidRecordId);

Log copy(out LogId id);

```

---

```
    Log\_copy\_with\_id(in LogId id) raises(LogIdAlreadyExists);

    void flush() raises(UnsupportedQoS);
};
```

### **Log::my\_factory()**

```
LogMgr my_factory();
```

Returns an object reference to the log object's log factory.

### **Log::id()**

```
LogId id();
```

Returns the id of the log.

### **Log::get\_max\_record\_life()**

```
unsigned long get_max_record_life();
```

Returns the maximum amount of time, in seconds, that a record stays valid in the log.

### **Log::set\_max\_record\_life()**

```
void set_max_record_life(in unsigned long life);
```

Sets the maximum amount of time, in seconds, that a record stays valid in the log. After a record has become stale, it will automatically be removed from the log.

#### **Parameters**

<code>life</code>	The number of seconds for which records will remain valid. Zero specifies an infinite life span.
-------------------	--

---

### **Log::get\_max\_size()**

```
unsigned long long get_max_size();
```

Returns the maximum size, in bytes, of the log.

### **Log::set\_max\_size()**

```
void set_max_size(in unsigned long long size)  
    raises(InvalidParam);
```

Set the maximum size, in bytes, of the log.

#### **Parameters**

size                    The maximum size of the log object in bytes.

#### **Exceptions**

[InvalidParam](#)    The size specified is smaller than the current size of the log.

### **Log::get\_current\_size()**

```
unsigned long long get_current_size();
```

Returns the current size of the log in octets.

### **Log::get\_n\_records()**

```
unsigned long long get_n_records();
```

Returns the current number of records in the log.

### **Log::get\_log\_full\_action()**

```
LogFullActionType get_log_full_action();
```

Returns the log's `full_action` setting.

---

## Log::set\_log\_full\_action()

```
void set_log_full_action(in LogFullActionType action)
raises(InvalidLogFullAction);
```

Sets the log's full\_action.

### Parameters

action            The log's full\_action. Valid values are wrap and halt.

### Exceptions

[InvalidLogFullAction](#) The full\_action specified is not a supported.

## Log::get\_administrative\_state()

```
AdministrativeState get_administrative_state();
```

Returns the log's administrative state.

## Log::set\_administrative\_state()

```
void set_administrative_state(in AdministrativeState state);
```

Sets the log's administrative state.

### Parameters

state            The new administrative state for the log. Valid states are locked and unlocked.

## Log::get\_forwarding\_state()

```
ForwardingState get_forwarding_state();
```

Returns the log's forwarding state. If the log's forwarding state is on, the log will forward events.

---

## Log::set\_forwarding\_state()

```
void set_forwarding_state(in ForwardingState state);
```

Changes the log's forwarding state.

### Parameters

state	The new forwarding state. The valid values are: on—specifies that the log will forward events. off—specifies that the log will not forward events.
-------	--

## Log::get\_operational\_state()

```
OperationalState get_operational_state();
```

Returns the log's operational state. The log can either be enabled or disabled.

## Log::get\_interval()

```
TimeInterval get_interval();
```

Returns the log's coarse grained logging interval.

## Log::set\_interval()

```
void set_interval(in TimeInterval interval)  
raises (InvalidTime, InvalidTimeInterval);
```

Changes the log's coarse grained logging interval.

### Parameters

interval	The log's new coarse grained logging interval. Zero sets the log to an infinite duration.
----------	---

---

## Exceptions

[InvalidTime](#) One of the times specified is not a legal time.  
[InvalidTimeInterval](#) The start time of the interval is after the stop time.  
Also, the stop time is prior to the current time.

### **Log::get\_availability\_status()**

[AvailabilityStatus](#) `get_availability_status();`

Returns the log's availability. The log can be on duty, off duty, full, or both off duty and full.

### **Log::get\_capacity\_alarm\_thresholds()**

[CapacityAlarmThresholdList](#) `get_capacity_alarm_thresholds();`

Returns a list of the log's alarm thresholds.

### **Log::set\_capacity\_alarm\_thresholds()**

`void set_capacity_alarm_thresholds(in CapacityAlarmThresholdList threshs)`  
`raises (InvalidThreshold);`

Sets threshold alarms in the log.

## Parameters

`threshs` A sequence of [Threshold](#) specifying at what points threshold alarm events are to be generated.

## Exceptions

[InvalidThreshold](#) Raised if one of the thresholds is not in the valid range.

---

## Log::get\_week\_mask()

[WeekMask](#) get\_week\_mask();

Returns the log's weekly schedule.

## Log::set\_week\_mask()

void set\_week\_mask(in [WeekMask](#) masks)  
raises ([InvalidTime](#), [InvalidTimeInterval](#), [InvalidMask](#));

Changes the log's weekly schedule.

### Parameters

masks            The new schedule to set on the log.

### Exceptions

[InvalidTime](#)            One of the times set on the log is not a valid time.

[InvalidTimeInterval](#) One of the stop times specified is before its associated start time. Also, one of the time intervals overlaps another time interval.

[InvalidMask](#)            The [WeekMask](#) is malformed.

## Log::get\_log\_qos()

[QoSList](#) get\_log\_qos();

Returns the log's QoS settings.

## Log::set\_log\_qos()

void set\_log\_qos(in [QoSList](#) qos) raises ([UnsupportedQoS](#));

Sets the log's QoS type. Valid settings are QoSNone, QoSFlush, and QoSReliable.

### Parameters

qos                The QoS properties to set on the log.



---

## Exceptions

[UnsupportedQoS](#) One of the QoS properties specified for the log is invalid. The invalid setting is returned.

## Log::query()

```
RecordList query(in string grammar, in Constraint c, out Iterator i)
raises(InvalidGrammar, InvalidConstraint);
```

Retrieves records from the log based on a constraint.

## Parameters

grammar	The grammar used to construct the constraint. The telecom logging service support the EXTENDED_TCL grammar
c	The constraint string against which records are matched.
i	Used when a large number of records are retrieved. If it not used it will be nil.

## Exceptions

[InvalidGrammar](#) The telecom logging service does not support the specified grammar.

[InvalidConstraint](#) The constraint does not conform to the specified grammar.

## Log::retrieve()

```
RecordList retrieve(in TimeT from_time, in long how_many,
out Iterator i);
```

Returns the specified number of records starting at the specified time. If the number of records is larger than can be stored in the return parameter, the remaining records are accessible through the [Iterator](#).

---

## Parameters

<code>from_time</code>	The time at which the first record to retrieve was logged.
<code>how_many</code>	The number of records to retrieve. A negative value causes the method to retrieve records prior to the specified time.
<code>i</code>	The <a href="#">Iterator</a> object reference.

## **Log::match()**

```
unsigned long match(in string grammar, in Constraint c)
raises(InvalidGrammar, InvalidConstraint);
```

Returns the number of records that match the specified constraint.

## Parameters

<code>grammar</code>	The grammar used to specify the constraint. The telecom logging service supports the <code>EXTENDED_TCL</code> grammar.
<code>c</code>	The constraint string.

## Exceptions

[InvalidGrammar](#) The telecom logging service does not support the specified grammar.

[InvalidConstraint](#) The constraint does not conform to the specified grammar.

## **Log::delete\_records()**

```
unsigned long delete_records(in string grammar, in Constraint c)
raises(InvalidGrammar, InvalidConstraint);
```

Deletes all of the records that match the specified constraint and returns the number of records deleted.

---

## Parameters

<code>grammar</code>	The grammar used to specify the constraint. The telecom logging service supports the <code>EXTENDED_TCL</code> grammar.
<code>c</code>	The constraint string.

## Exceptions

<a href="#"><u>InvalidGrammar</u></a>	The telecom logging service does not support the specified grammar.
<a href="#"><u>InvalidConstraint</u></a>	The constraint does not conform to the specified grammar.

## **Log::delete\_records\_by\_id()**

```
unsigned long delete_records_by_id(in RecordIdList ids);
```

Deletes the specified records and returns the number of deleted records.

## Parameters

<code>ids</code>	A sequence of record ids specifying the records to delete.
------------------	--

## **Log::write\_records()**

```
void write_records(in Anys records)  
raises(LogFull, LogOffDuty, LogLocked, LogDisabled);
```

Writes a series of records to a log. The you cannot specify any optional attributes and cannot discover the records id.

## Parameters

<code>records</code>	A sequence of <code>any</code> that contains the data for a group of records.
----------------------	---

## Exceptions

<a href="#"><u>LogFull</u></a>	The log is full and its <code>full_action</code> is set to <code>halt</code> .
<a href="#"><u>LogOffDuty</u></a>	The log is not currently scheduled to accept new records.

- 
- |                             |   |
|-----------------------------|---|
| <a href="#">LogLocked</a>   | The log's administrative state is set to not accept new records.                |
| <a href="#">LogDisabled</a> | The log has encountered a processing error and is unable to accept new records. |

### **Log::write\_recordlist()**

```
void write_recordlist(in RecordList list)
raises(LogFull, LogOffDuty, LogLocked, LogDisabled);
```

Writes a series of records to the log. You can construct records that include an optional attribute list and each record in the list will be updated to include the time it was logged and its record id.

#### **Parameters**

- |      |  |
|------|--|
| list | A sequence of <a href="#">LogRecord</a> that contains the data for a group of records. |
|------|--|

#### **Exceptions**

- |                             |   |
|-----------------------------|---|
| <a href="#">LogFull</a>     | The log is full and its <code>full_action</code> is set to <code>halt</code> .  |
| <a href="#">LogOffDuty</a>  | The log is not currently scheduled to accept new records.                       |
| <a href="#">LogLocked</a>   | The log's administrative state is set to not accept new records.                |
| <a href="#">LogDisabled</a> | The log has encountered a processing error and is unable to accept new records. |

### **Log::set\_record\_attribute()**

```
void set_record_attribute(in RecordId id, in NVList attr_list)
raises(InvalidRecordId, InvalidAttribute);
```

Sets attributes for a single record which is specified by its record id.

#### **Parameters**

- |           |  |
|-----------|--|
| id        | The id of the record on which you wish to set attributes.  |
| attr_list | The list of attributes that you want to set on the record. |

---

## Exceptions

[InvalidRecordId](#) The record specified does not exist.

[InvalidAttribute](#) One of the attributes is illegal.

## Log::set\_records\_attribute()

```
unsigned long set_records_attribute(in string grammar,
                                  in Constraint c,
                                  in NVList attr_list)
raises(InvalidGrammar, InvalidConstraint, InvalidAttribute);
```

Sets attributes for all records that match the constraint. It returns the numbers of records whose attributes were changed.

## Parameters

grammar	The grammar used to specify the constraint. The telecom logging service supports the <code>EXTENDED_TCL</code> grammar.
c	The constraint string.
attr_list	The list of attributes that you want to set on the record.

## Exceptions

[InvalidGrammar](#) The telecom logging service does not support the specified grammar.

[InvalidConstraint](#) The constraint does not conform to the specified grammar.

[InvalidAttribute](#) One of the attributes is illegal.

## Log::get\_record\_attribute()

```
NVList get_record_attribute(in RecordId id)
raises(InvalidRecordId);
```

Returns the list of attributes that are set on the specified record.

---

## Parameters

id                    The id of the record whose attributes you want to retrieve.

## Exceptions

[InvalidRecordId](#) The record specified does not exist.

## Log::copy()

```
Log copy(out LogId id);
```

Copies the log object and returns a reference to the new log object.

## Parameters

id                    The id assigned to the newly created log.

## Log::copy\_with\_id()

```
Log copy_with_id(in LogId id)  
raises (LogIdAlreadyExists);
```

Copies the log and returns a reference to the newly created log. This method allows you to specify the logs id.

## Parameters

id                    The new log's id.

## Exceptions

[LogIdAlreadyExists](#)The user assigned id is already in use.

## Log::flush()

```
void flush()  
raises(UnsupportedQoS);
```

Causes the log to flush its memory buffer to its associated permanent store.

---

## Exceptions

[UnsupportedQoS](#) The log does not support QoSFlush.





# DsLogAdmin::LogMgr Interface

The `LogMgr` interface is inherited by all the log factory interfaces. It defines three methods of discovering deployed log objects.

```
interface LogMgr
{
  LogList list_logs();
  Log find_log(in LogId id);
  LogIdList list_logs_by_id();
};
```

## **LogMgr::list\_logs()**

```
LogList list_logs();
```

Returns a list of object references, one for each log object associated with the factory.

## **LogMgr::find\_log()**

```
Log find_log(in LogId id);
```

Returns an object reference to the specified log. If the log does not exist, it returns a `nil` reference.

## **LogMgr::list\_logs\_by\_id()**

```
LogIdList list_logs_by_id();
```

Returns a list containing the ids of all logs associated with the factory.



# DsLogNotification Module

The `DsLogNotification` module defines the data types used to transmit log generated events to logging clients.

## DsLogNotification::PerceivedSeverityType Type

```
typedef unsigned short PerceivedSeverityType;
const PerceivedSeverityType critical = 0;
const PerceivedSeverityType minor = 1;
const PerceivedSeverityType cleared = 2;
```

Defines the severity of a threshold alarm. A threshold alarm's severity is considered `minor` unless the log is full.

## DsLogNotification::ThresholdAlarm Structure

```
struct ThresholdAlarm
{
    Log          logref;
    LogId       id;
    TimeT       time;
    Threshold   crossed_value;
    Threshold   observed_value;
    PerceivedSeverityType perceived_severity;
};
```

The data type passed in a threshold alarm event.

<code>logref</code>	An object reference to the log object which caused the event.
<code>id</code>	The id of the log object which caused the event.
<code>time</code>	The time the event was generated.
<code>crossed_value</code>	The capacity threshold which was passed to trigger the event.

---

observed\_value      The actual percentage of the log that is full.  
perceived\_severity The severity of the alarm. If the severity is critical then  
the log object is full.

## DsLogNotification::ObjectCreation Structure

```
struct ObjectCreation
{
    LogId id;
    TimeT time;
};
```

The data type passed in an object creation event.

id                      The id of the newly created log object.  
time                    The time the log object was generated.

## DsLogNotification::ObjectDeletion Structure

```
struct ObjectDeletion
{
    LogId id;
    TimeT time;
};
```

The data type passed in an object deletion event.

id                      The id of the deleted log object.  
time                    The time the log object was deleted.

## DsLogNotification::AttributeType Type

```
typedef unsigned short AttributeType;
const AttributeType capacityAlarmThreshold = 0;
const AttributeType logFullAction        = 1;
const AttributeType maxLogSize           = 2;
const AttributeType startTime            = 3;
```

---

```
const AttributeType stopTime           = 4;
const AttributeType weekMask          = 5;
const AttributeType filter             = 6;
const AttributeType maxRecordLife     = 7;
const AttributeType qualityOfService  = 8;
```

The data type and constants used to represent the type of attribute changed in an attribute change event.

## DsLogNotification::AttributeValueChange Structure

```
struct AttributeValueChange
{
    Log logref;
    LogId id;
    TimeT time;
    AttributeType type;
    any old_value;
    any new_value;
};
```

logref An object reference to the log object which caused the event.

id The id of the log object which caused the event.

time The time the event was generated.

type The attribute that was changed.

old\_value The previous value of the attribute.

new\_value The attribute's new value.

## DsLogNotification::StateType Type

```
typedef unsigned short StateType;
const StateType administrativeState = 0;
const StateType operationalState    = 1;
const StateType forwardingState     = 2;
```

The data type and constants used to represent which type of state was changed in a state change event.

---

## DsLogNotification::StateChange Structure

```
struct StateChange
{
    Log logref;
    LogId id;
    TimeT time;
    StateType type;
    any new_value;
};
```

The data type passed in a state change event.

logref An object reference to the log object which caused the event.

id The id of the log object which caused the event.

time The time the event was generated.

type The type of state that was changed.

new\_value The new state.

## DsLogNotification::ProcessingErrorAlarm Structure

```
struct ProcessingErrorAlarm
{
    long error_num;
    string error_string;
};
```

The data type passed when a processing error event occurs.

error\_num The error number.

error\_string A string explaining the error.

# DsNotifyLogAdmin Module

The `DsNotifyLogAdmin` module extends the functionality of the interfaces specified in the [DsLogAdmin](#) module to support notification style push and pull communication and forwarding of structured and sequenced events. The extended functionality also includes notification style event filtering and subscription/publication functionality.





# DsNotifyLogAdmin::NotifyLog Interface

The `NotifyLog` interface extends the functionality of the [Log](#) interface to support notification style filters. It inherits from the [EventChannel](#) interface of module [CosNotifyChannelAdmin](#).

```
interface NotifyLog : DsEventLogAdmin::EventLog,  
                    CosNotifyChannelAdmin::EventChannel  
{  
    CosNotifyFilter::Filter get_filter();  
    void set_filter(in CosNotifyFilter::Filter filter);  
};
```

## NotifyLog::get\_filter()

```
CosNotifyFilter::Filter get_filter();
```

Returns a reference to the filter object associated with the log.

## NotifyLog::set\_filter()

```
void set_filter(in CosNotifyFilter::Filter filter);
```

Associates a filter with the log. The filter will determine which events will be logged.

### Parameters

<code>filter</code>	The filter you want to set on the log.
---------------------	--



# DsNotifyLogAdmin::NotifyLogFactory Interface

The `NotifyLogFactory` extends the functionality of the [LogMgr](#) interface to support the creation of [NotifyLog](#) objects. It also inherits from the [CosNotifyChannelAdmin::ConsumerAdmin](#) interface. This inheritance allows it to forward events to the clients of its associated [NotifyLog](#) objects.

## NotifyLogFactory::create()

```
NotifyLog create(in DsLogAdmin::LogFullActionType full_action,
                 in unsigned long long max_size,
                 in DsLogAdmin::CapacityAlarmThresholdList thresholds,
                 in CosNotification::QoSProperties initial_qos,
                 in CosNotification::AdminProperties initial_admin,
                 out DsLogAdmin::LogId id)
raises(DsLogAdmin::InvalidLogFullAction,
       DsLogAdmin::InvalidThreshold,
       CosNotification::UnsupportedQoS,
       CosNotification::UnsupportedAdmin);
```

Creates a new [NotifyLog](#) object, assigns the new log a unique id, and returns a reference to the newly instantiated log object.

### Parameters

<code>full_action</code>	The log's behavior when it reaches its maximum size. Valid values are <code>wrap</code> and <code>halt</code> .
<code>max_size</code>	The maximum size of the log in bytes.
<code>thresholds</code>	The thresholds when alarm events will be generated. Specified as a percentage of the log's size.
<code>initial_qos</code>	The initial notification style QoS properties to set on the log object's associated notification channel.

---

`initial_admin` The initial administrative properties to set on the log object's associated notification channel.

`id` Returns the log object's factory assigned id.

## Exceptions

[InvalidLogFullAction](#) The value for the log's `full_action` was not a valid `full_action`.

[InvalidThreshold](#) One of the threshold alarm values was not within the valid range

[UnsupportedQoS](#) One of the QoS properties is invalid or does not support the value you are trying to set for it.

[UnsupportedAdmin](#) One of the administrative properties is invalid or does not support the value you are trying to set for it.

## NotifyLogFactory::create\_with\_id()

```
NotifyLog create_with_id(in DsLogAdmin::LogId id,  
    in DsLogAdmin::LogFullActionType full_action,  
    in unsigned long long max_size,  
    in DsLogAdmin::CapacityAlarmThresholdList thresholds,  
    in CosNotification::QoSProperties initial_qos,  
    in CosNotification::AdminProperties initial_admin)  
raises(DsLogAdmin::LogIdAlreadyExists,  
    DsLogAdmin::InvalidLogFullAction,  
    DsLogAdmin::InvalidThreshold,  
    CosNotification::UnsupportedQoS,  
    CosNotification::UnsupportedAdmin);
```

Creates a new [NotifyLog](#) object using a user assigned id and returns a reference to the newly instantiated log object.

## Parameters

`id` The log object's id.

`full_action` The log's behavior when it reaches its maximum size. Valid values are `wrap` and `halt`.

`max_size` The maximum size of the log in bytes.

---

<code>thresholds</code>	The thresholds when alarm events will be generated. Specified as a percentage of the log's size.
<code>initial_qos</code>	The initial notification style QoS properties to set on the log object's associated notification channel.
<code>initial_admin</code>	The initial administrative properties to set on the log object's associated notification channel.

## Exceptions

<a href="#"><u>LogIdAlreadyExists</u></a>	A log already exists with the specified id.
<a href="#"><u>InvalidLogFullAction</u></a>	The value for the log's <code>full_action</code> was not a valid <code>full_action</code> .
<a href="#"><u>InvalidThreshold</u></a>	One of the threshold alarm values was not within the valid range
<a href="#"><u>UnsupportedQoS</u></a>	One of the QoS properties is invalid or does not support the value you are trying to set for it.
<a href="#"><u>UnsupportedAdmin</u></a>	One of the administrative properties is invalid or does not support the value you are trying to set for it.



# Dynamic Module

The `Dynamic` module is used by the `PortableInterceptor` module and contains the following data types:

- [ContextList](#) type
- [ExceptionList](#) sequence
- [Parameter](#) structure
- [ParameterList](#) sequence
- [RequestContext](#) type

## Dynamic::ContextList

```
// IDL
typedef CORBA::StringSeq ContextList;
```

## Dynamic::ExceptionList

```
// IDL
typedef sequence<CORBA::TypeCode> ExceptionList;
```

## Dynamic::Parameter

```
// IDL
struct Parameter {
    any argument;
    CORBA::ParameterMode mode;
};
```

### **Dynamic::ParameterList**

```
// IDL
typedef sequence<Parameter> ParameterList;
```

### **Dynamic::RequestContext**

```
// IDL
typedef CORBA::StringSeq RequestContext;
```



# DynamicAny Overview

The `DynamicAny` namespace implements the IDL `DynamicAny` module which includes the following classes:

[DynAny](#)  
[DynAnyFactory](#)  
[DynArray](#)  
[DynEnum](#)  
[DynFixed](#)  
[DynSequence](#)  
[DynStruct](#)  
[DynUnion](#)  
[DynValue](#)

The common data types in the scope of the `DynamicAny` module include the following:

[NameDynAnyPair](#)  
[NameValuePair](#)

For most IDL data types there is a straight-forward language mapping that an object implementation uses to interpret data. However, an `any` data type can be passed to a program that may not have any static information about how to interpret the type of data in the `any` value. The `DynamicAny` module provides a runtime mechanism for constructing `any` values, traversing them, and extracting the data from `any` values. This mechanism is especially helpful for writing generic clients and servers such as bridges, browsers, debuggers, and user interface tools.

Applications dynamically construct and interpret `any` values using [DynAny](#) objects. For complex `any` types a [DynAny](#) object is an ordered collection of other component [DynAny](#) objects.

A [DynAny](#) object can be created as follows:

- Invoking a method on a [DynAnyFactory](#) object.
- Invoking a method on an existing [DynAny](#) object.

A constructed [DynAny](#) object supports methods that enable the creation of new [DynAny](#) objects that encapsulate access to the value of some

---

constituent of the [DynAny](#) object. [DynAny](#) objects also support a copy method for creating new [DynAny](#) objects.

There is a different interface associated with each kind of constructed IDL type that inherits from the [DynAny](#) interface. The interfaces that inherit the [DynAny](#) interface include:

[DynArray](#)  
[DynEnum](#)  
[DynFixed](#)  
[DynSequence](#)  
[DynStruct](#)  
[DynUnion](#)  
[DynValue](#)

Exceptions are represented by the [DynStruct](#) interface and value types are represented by the [DynValue](#) interface.

A sequence of [CORBA::Any](#) values.

#### See Also

[DynamicAny::DynSequence](#)  
[DynamicAny::DynArray](#)

## DynamicAny::NameDynAnyPair Structure

```
// IDL
struct NameDynAnyPair {
    FieldName id;
    DynAny value;
};

// Java
public final class NameDynAnyPair
    implements org.omg.CORBA.portable.IDLEntity
{
    public java.lang.String id;
    public org.omg.DynamicAny.DynAny value;

    public NameDynAnyPair() {}
    public NameDynAnyPair(
        java.lang.String id,
        org.omg.DynamicAny.DynAny value
    )
    {
```

---

```
        this.id = id;
        this.value = value;
    }
}
```

A structure containing the name and value of a field or member.

### **DynamicAny::NameValuePair Structure**

```
// Java
public final class NameValuePair
    implements org.omg.CORBA.portable.IDLEntity
{
    public java.lang.String id;
    public org.omg.CORBA.Any value;

    public NameValuePair() {}
    public NameValuePair(
        java.lang.String id,
        org.omg.CORBA.Any value
    )
    {
        this.id = id;
        this.value = value;
    }
}
```

A structure containing the name and value of a field or member.



# DynamicAny::DynAny Class

Your application can dynamically construct and interpreted `Any` values using `DynAny` objects. A `DynAny` object is associated with a data value which corresponds to a copy of the value inserted into an `any`. Portable programs should use the `DynAny` interface to access and modify the contents of an `Any` in those cases where basic insertion and extraction operators are not sufficient.

`DynAny` methods can be organized as follows:

**Table 11:** *DynAny Methods*

General Methods	Insert Methods	Get Methods
<a href="#"><u>assign()</u></a>	<a href="#"><u>insert_any()</u></a>	<a href="#"><u>get_any()</u></a>
<a href="#"><u>component_count()</u></a>	<a href="#"><u>insert_boolean()</u></a>	<a href="#"><u>get_boolean()</u></a>
<a href="#"><u>copy()</u></a>	<a href="#"><u>insert_char()</u></a>	<a href="#"><u>get_char()</u></a>
<a href="#"><u>current_component()</u></a>	<a href="#"><u>insert_double()</u></a>	<a href="#"><u>get_double()</u></a>
<a href="#"><u>destroy()</u></a>	<a href="#"><u>insert_dyn_any()</u></a>	<a href="#"><u>get_dyn_any()</u></a>
<a href="#"><u>equal()</u></a>	<a href="#"><u>insert_float()</u></a>	<a href="#"><u>get_float()</u></a>
<a href="#"><u>from_any()</u></a>	<a href="#"><u>insert_long()</u></a>	<a href="#"><u>get_long()</u></a>
<a href="#"><u>next()</u></a>	<a href="#"><u>insert_longlong()</u></a>	<a href="#"><u>get_longlong()</u></a>
<a href="#"><u>rewind()</u></a>	<a href="#"><u>insert_octet()</u></a>	<a href="#"><u>get_octet()</u></a>
<a href="#"><u>seek()</u></a>	<a href="#"><u>insert_reference()</u></a>	<a href="#"><u>get_reference()</u></a>
<a href="#"><u>to_any()</u></a>	<a href="#"><u>insert_short()</u></a>	<a href="#"><u>get_short()</u></a>
<a href="#"><u>type()</u></a>	<a href="#"><u>insert_string()</u></a>	<a href="#"><u>get_string()</u></a>
	<a href="#"><u>insert_typecode()</u></a>	<a href="#"><u>get_typecode()</u></a>
	<a href="#"><u>insert_ulong()</u></a>	<a href="#"><u>get_ulong()</u></a>
	<a href="#"><u>insert_ulonglong()</u></a>	<a href="#"><u>get_ulonglong()</u></a>
	<a href="#"><u>insert_ushort()</u></a>	<a href="#"><u>get_ushort()</u></a>
	<a href="#"><u>insert_val()</u></a>	<a href="#"><u>get_val()</u></a>
	<a href="#"><u>insert_wchar()</u></a>	<a href="#"><u>get_wchar()</u></a>
	<a href="#"><u>insert_wstring()</u></a>	<a href="#"><u>get_wstring()</u></a>

The following exceptions are also defined in the `DynAny` class:

[InvalidValue](#)

[TypeMismatch](#)

---

The `DynAny` class is the base for the following classes:

[DynArray](#)

[DynEnum](#)

[DynFixed](#)

[DynSequence](#)

[DynStruct](#)

[DynUnion](#)

[DynValue](#)

Because the values of `Any` types can be quite complex, it is helpful to think of a `DynAny` object as an ordered collection of other *component* `DynAny` objects. For simpler `DynAny` objects that represent a basic type, the ordered collection of components is empty. For example, a `long` or a type without components (such as an empty exception) has empty components.

The `DynAny` interface allows a client to iterate through the components of the values pointed to by these objects. Each `DynAny` object maintains the notion of a *current position* into its collection of component `DynAny` objects. The current position is identified by an index value that runs from 0 to  $n-1$ , where  $n$  is the number of components. Methods are available that allow you to recursively examine `DynAny` contents. For example, you can determine the current position using [current\\_component\(\)](#), and [component\\_count\(\)](#) returns the number of components in the `DynAny` object. You can also use [rewind\(\)](#), [seek\(\)](#), and [next\(\)](#) to change the current position. If a `DynAny` is initialized with a value that has components, the index is initialized to 0. The special index value of -1 indicates a current position that points nowhere. For example, some values (such as an empty exception) cannot have a current position. In these cases the index value is fixed at -1.

You can use the iteration operations, together with [current\\_component\(\)](#), to dynamically compose an `Any` value. After creating a dynamic any, such as a `DynStruct`, you can use [current\\_component\(\)](#) and [next\(\)](#) to initialize all the components of the value. Once the dynamic value is completely initialized, [to\\_any\(\)](#) creates the corresponding `Any` value.

You use the [insert\\_type\(\)](#) and [get\\_type\(\)](#) methods to not only handle basic `DynAny` objects but they are also helpful in handling constructed `DynAny` objects. when you insert a basic data type value into a constructed `DynAny` object, it initializes the current component of the constructed data value associated with the `DynAny` object.

---

For example, invoking [insert\\_boolean\(\)](#) on a `DynStruct` object implies inserting a boolean data value at the current position of the associated structure data value. In addition, you can use the `insert_type()` and `get_type()` methods to traverse `Any` values associated with sequences of basic data types without the need to generate a `DynAny` object for each element in the sequence.

The `DynAny` object has a [destroy\(\)](#) method that you can use to destroy a top-level `DynAny` object and any component `DynAny` objects obtained from it.

## Exceptions

`TypeMismatch` is raised if you call methods `insert_type()` or `get_type()` on a `DynAny` whose current component itself has components.

`MARSHAL` is raised if you attempt to export `DynAny` objects to other processes or externalize one with [CORBA::ORB::object\\_to\\_string\(\)](#). This is because `DynAny` objects are intended to be local to the process in which they are created and used.

`NO_IMPLEMENT` might be raised if you attempt the following:

- Invoke operations exported through the [CORBA::Object](#) interface even though `DynAny` objects export operations defined in this standard interface.
- Use a `DynAny` object with the DII.

The following code is the complete class:

```
package org.omg.DynamicAny;

public interface DynAny extends org.omg.CORBA.Object
{
    org.omg.CORBA.TypeCode type();

    void assign(
        org.omg.DynamicAny.DynAny dyn_any
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    void from_any(
        org.omg.CORBA.Any value
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
        org.omg.DynamicAny.DynAnyPackage.InvalidValue;

    org.omg.CORBA.Any to_any();
}
```

---

```
boolean equal(
    org.omg.DynamicAny.DynAny dyn_any
);

void destroy();

org.omg.DynamicAny.DynAny copy();

void insert_boolean(
    boolean value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_octet(
    byte value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_char(
    char value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_short(
    short value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_ushort(
    short value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_long(
    int value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_ulong(
    int value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_float(
    float value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_double(
    double value
```



---

```
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_string(
    java.lang.String value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_reference(
    org.omg.CORBA.Object value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_typecode(
    org.omg.CORBA.TypeCode value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_longlong(
    long value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_ulonglong(
    long value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_wchar(
    char value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_wstring(
    java.lang.String value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_any(
    org.omg.CORBA.Any value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_dyn_any(
    org.omg.DynamicAny.DynAny value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

void insert_val(
    java.io.Serializable value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

---

```
    boolean get_boolean() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    byte get_octet() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    char get_char() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    short get_short() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    short get_ushort() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    int get_long() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    int get_ulong() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    float get_float() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    double get_double() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    java.lang.String get_string() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    org.omg.CORBA.Object get_reference() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    org.omg.CORBA.TypeCode get_typecode() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    long get_longlong() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    long get_ulonglong() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;
```

---

```
    char get_wchar() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    java.lang.String get_wstring() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    org.omg.CORBA.Any get_any() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    org.omg.DynamicAny.DynAny get_dyn_any() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    java.io.Serializable get_val() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    boolean seek(
        int index
    );

    void rewind();

    boolean next();

    int component_count() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    org.omg.DynamicAny.DynAny current_component() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;
}
```

## **DynAny::assign()**

```
// Java
void assign(
    org.omg.DynamicAny.DynAny dyn_any
)
```

Initializes the value associated with a `DynAny` object with the value associated with another `DynAny` object.

---

## Parameters

`dyn_any`            The `DynAny` object to initialize to.

The current position of the target `DynAny` is set to zero for values that have components and to -1 for values that do not have components.

## Exceptions

[TypeMismatch](#)    The type of the passed `DynAny` is not equivalent to the type of the target `DynAny`.

## `DynAny::component_count()`

```
int component_count()
```

Returns the number of components of a `DynAny`. For a `DynAny` without components, it returns zero.

The operation only counts the components at the top level. For example, if you invoke `component_count()` on a `DynStruct` with a single member, the return value is 1, irrespective of the type of the member.

**Table 12:** *Return Values for `DynAny::component_count()`*

Type	Return Value
<a href="#">DynSequence</a>	The current number of elements.
<a href="#">DynStruct</a> <a href="#">DynValue</a>	The number of members.
<a href="#">DynArray</a>	The number of elements.
<a href="#">DynUnion</a>	2 if the discriminator indicates that a named member is active. 1 Otherwise.
<a href="#">DynFixed</a> <a href="#">DynEnum</a>	zero

---

## Exceptions

`TypeMismatch` The method is called on a `DynAny` that cannot have components, such as a `DynEnum` or an empty exception.

## See Also

```
DynamicAny::DynAny::current_component()  
DynamicAny::DynAny::seek()  
DynamicAny::DynAny::rewind()  
DynamicAny::DynAny::next()
```

## DynAny::copy()

```
org.omg.DynamicAny.DynAny copy();
```

Returns a new `DynAny` object whose value is a deep copy of the `DynAny` on which it is invoked.

The operation is polymorphic, that is, invoking it on one of the types derived from `DynAny`, such as [DynStruct](#), creates the derived type but returns its reference as the `DynAny` base type.

## DynAny::current\_component()

```
// Java  
org.omg.DynamicAny.DynAny current_component();
```

Returns the `DynAny` for the component at the current position. It does not advance the current position, so repeated calls without an intervening call to `rewind()`, `next()`, or `seek()` return the same component. If the current position current position is `-1`, the method returns a `nil` reference.

The returned `DynAny` object reference can be used to get or set the value of the current component. If the current component represents a complex type, the returned reference can be narrowed based on the `TypeCode` to get the interface corresponding to the complex type.

## Exceptions

`TypeMismatch` The method is called on a `DynAny` that cannot have components, such as a `DynEnum` or an empty exception.

## See Also

```
DynamicAny::DynAny::component_count()  
DynamicAny::DynAny::seek()
```

---

```
DynamicAny::DynAny::rewind()  
DynamicAny::DynAny::next()
```

## DynAny::destroy()

```
// Java  
void destroy();
```

Destroys a `DynAny` object. This operation frees any resources used to represent the data value associated with a `DynAny` object.

Destroying a top-level `DynAny` object (one that was not obtained as a component of another `DynAny`) also destroys any component `DynAny` objects obtained from it. Destroying a non-top level (component) `DynAny` object does nothing.

You can manipulate a component of a `DynAny` object beyond the life time of its top-level `DynAny` by making a copy of the component with [copy\(\)](#) before destroying the top-level `DynAny` object.

**Enhancement** Orbix guarantees to always destroy all local objects it creates when the last reference to them is released so you do not have to call `destroy()`. However, code that relies on this feature is not strictly CORBA compliant and may leak resources with other ORBs.

### Exceptions

`OBJECT_NOT_EXIST` A destroyed `DynAny` object or any of its components is referenced.

### See Also

[DynamicAny::DynAny::copy\(\)](#)  
`IT_CORBA::RefCountedLocalObject`

## DynAny::equal()

```
boolean equal(  
    org.omg.DynamicAny.DynAny dyn_any  
);
```

Compares two `DynAny` values for equality and returns true if the values are equal, false otherwise. Two `DynAny` values are equal if their type codes are equivalent and, recursively, all respective component `DynAny` values are equal.

---

The current position of the two `DynAny` values being compared has no effect on the result of `equal()`.

### Parameters

`dyn_any`            The `DynAny` value to compare.

### `DynAny::from_any()`

```
// Java
void from_any(
    org.omg.CORBA.Any value
);
```

Initializes the value associated with a `DynAny` object with the value contained in an [Any](#) type.

The current position of the target `DynAny` is set to zero for values that have components and to -1 for values that do not have components.

### Parameters

`value`            An [Any](#) value to initialize the `DynAny` object to.

### Exceptions

[TypeMismatch](#)    The type of the passed `Any` is not equivalent to the type of the target `DynAny`.

[InvalidValue](#)    The passed `Any` does not contain a legal value (such as a null string).

### See Also

[DynamicAny::DynAny::to\\_any\(\)](#)

### `DynAny::get_any()`

```
org.omg.CORBA.Any get_any();
```

Returns an [Any](#) value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_any` (an [Any TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_any`. The current position is unchanged after the call.

---

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::insert\\_any\(\)](#)

## DynAny::get\_boolean()

```
boolean get_boolean();
```

Returns a `boolean` value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_boolean` (a `boolean` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_boolean`. The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::insert\\_boolean\(\)](#)

## DynAny::get\_char()

```
char get_char();
```

Returns a `char` value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_char` (a `char` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_char`. The current position is unchanged after the call.



---

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::insert\\_char\(\)](#)

## DynAny::get\_double()

```
double get_double();
```

Returns a `double` value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_double` (a `double` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_double`. The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::insert\\_double\(\)](#)

## DynAny::get\_dyn\_any()

```
org.omg.DynamicAny.DynAny get_dyn_any();
```

Returns a `DynAny` reference value from the `DynAny` object. `get_dyn_any()` is provided to deal with [Any](#) values that contain another any.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to the [TypeCode](#) of a `DynAny` or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent the [TypeCode](#) of a `DynAny`. The current position is unchanged after the call.

---

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::insert\\_dyn\\_any\(\)](#)

## `DynAny::get_float()`

```
float get_float();
```

Returns a `float` value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_float` (a `float` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_float`. The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::insert\\_float\(\)](#)

## `DynAny::get_long()`

```
int get_long();
```

Returns a `int` value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_long` (a `long` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_long`. The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

---

**See Also**      [DynamicAny::DynAny::insert\\_long\(\)](#)

## DynAny::get\_longlong()

```
long get_longlong();
```

Returns a `long` value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_longlong` (a `long long` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_longlong`. The current position is unchanged after the call.

### Exceptions

[TypeMismatch](#)    The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#)    The `DynAny` has components and the current position is -1.

**See Also**      [DynamicAny::DynAny::insert\\_longlong\(\)](#)

## DynAny::get\_octet()

```
byte get_octet();
```

Returns an `byte` value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_octet` (an `octet` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_octet`. The current position is unchanged after the call.

### Exceptions

[TypeMismatch](#)    The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#)    The `DynAny` has components and the current position is -1.

**See Also**      [DynamicAny::DynAny::insert\\_octet\(\)](#)

---

## DynAny::get\_reference()

```
org.omg.CORBA.Object get_reference();
```

Returns an [Object](#) reference from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_Object` (an object reference [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_Object`. The current position is unchanged after the call.

### Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

### See Also

[DynamicAny::DynAny::insert\\_reference\(\)](#)

## DynAny::get\_short()

```
short get_short();
```

Returns a `short` value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_short` (a `short` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_short`. The current position is unchanged after the call.

### Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

### See Also

[DynamicAny::DynAny::insert\\_short\(\)](#)

## DynAny::get\_string()

```
java.lang.String get_string();
```

Returns a string value from the `DynAny` object.

---

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_string` (a string [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_string`. The current position is unchanged after the call.

### Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

### See Also

[DynamicAny::DynAny::insert\\_string\(\)](#)

## DynAny::get\_typecode()

```
org.omg.CORBA.TypeCode get_typecode();
```

Returns a [TypeCode](#) value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_TypeCode` (a [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_TypeCode`. The current position is unchanged after the call.

### Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

### See Also

[DynamicAny::DynAny::insert\\_typecode\(\)](#)

## DynAny::get\_ulong()

```
int get_ulong();
```

Returns a `int` value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_ulong` (an unsigned long [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_ulong`. The current position is unchanged after the call.

---

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::insert\\_ulong\(\)](#)

## DynAny::get\_ulonglong()

```
long get_ulonglong();
```

Returns a `long` value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_ulonglong` (an unsigned `long long` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_ulonglong`. The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::insert\\_ulonglong\(\)](#)

## DynAny::get\_ushort()

```
short get_ushort();
```

Returns a `short` value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_ushort` (an unsigned `short` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_ushort`. The current position is unchanged after the call.

---

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::insert\\_ushort\(\)](#)

## DynAny::get\_val()

```
java.io.Serializable get_val();
```

Returns a value type value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to a value type [TypeCode](#), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to a value type [TypeCode](#). The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::insert\\_val\(\)](#)

## DynAny::get\_wchar()

```
char get_wchar();
```

Returns a `char` value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_wchar` (a `wchar` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_wchar`. The current position is unchanged after the call.

---

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::insert\\_wchar\(\)](#)

## DynAny::get\_wstring()

```
java.lang.String get_wstring();
```

Returns a wide string value from the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_wstring` (a wide string [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_wstring`. The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::insert\\_wstring\(\)](#)

## DynAny::insert\_any()

```
// Java
void insert_any(
    org.omg.CORBA.Any value
);
```

Inserts an [Any](#) value into the `DynAny` object.

## Parameters

`value` The value to insert into the `DynAny` object.



---

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_any` (an [Any TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_any`. The current position is unchanged after the call.

### Exceptions

[InvalidValue](#) The `DynAny` has components and the current position is -1.

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the inserted type.

### See Also

[DynamicAny::DynAny::get\\_any\(\)](#)

## DynAny::insert\_boolean()

```
// Java
void insert_boolean(
    boolean value
);
```

Inserts a `boolean` value into the `DynAny` object.

### Parameters

`value` The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_boolean` (a `boolean` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_boolean`. The current position is unchanged after the call.

### Exceptions

[InvalidValue](#) The `DynAny` has components and the current position is -1.

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the inserted type.

### See Also

[DynamicAny::DynAny::get\\_boolean\(\)](#)

---

## DynAny::insert\_char()

```
// Java
void insert_char(
    char value
);
```

Inserts a `char` value into the `DynAny` object.

### Parameters

`value`            The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_char` (a `char` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to [\\_tc\\_char](#). The current position is unchanged after the call.

### Exceptions

[InvalidValue](#)    The `DynAny` has components and the current position is -1.

[TypeMismatch](#)    The accessed component in the `DynAny` is of a type that is not equivalent to the inserted type.

### See Also

[DynamicAny::DynAny::get\\_char\(\)](#)

## DynAny::insert\_double()

```
// Java
void insert_double(double value);
```

Inserts a `double` value into the `DynAny` object.

### Parameters

`value`            The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_double` (a `double` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_double`. The current position is unchanged after the call.

---

## Exceptions

- [InvalidValue](#) The `DynAny` has components and the current position is -1.
- [TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the inserted type.

## See Also

[DynamicAny::DynAny::get\\_double\(\)](#)

## DynAny::insert\_dyn\_any()

```
// Java
void insert_dyn_any(
    org.omg.DynamicAny.DynAny value
);
```

Inserts a `DynAny` value into the `DynAny` object. `insert_dyn_any()` is provided to deal with [Any](#) values that contain another `any`.

## Parameters

`value` The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to the [TypeCode](#) of a `DynAny` or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent the [TypeCode](#) of a `DynAny`. The current position is unchanged after the call.

## Exceptions

- [InvalidValue](#) The `DynAny` has components and the current position is -1.
- [TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the inserted type.

## See Also

[DynamicAny::DynAny::get\\_dyn\\_any\(\)](#)

## DynAny::insert\_float()

```
// Java
void insert_float(float value);
```

Inserts a `float` value into the `DynAny` object.

---

## Parameters

value                    The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_float` (a `float` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_float`. The current position is unchanged after the call.

## Exceptions

[InvalidValue](#)        The `DynAny` has components and the current position is -1.

[TypeMismatch](#)        The accessed component in the `DynAny` is of a type that is not equivalent to the inserted type.

## See Also

[DynamicAny::DynAny::get\\_float\(\)](#)

## `DynAny::insert_long()`

```
// Java
void insert_long(
    int value
    ) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Inserts a `long` value into the `DynAny` object.

## Parameters

value                    The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_long` (a `long` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_long`. The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#)        The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#)        The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::get\\_long\(\)](#)

---

## DynAny::insert\_long long()

```
// Java
void insert_longlong(
    long value
);
```

Inserts a `long` value into the `DynAny` object.

### Parameters

`value`            The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_longlong` (a `long long` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_longlong`. The current position is unchanged after the call.

### Exceptions

[InvalidValue](#)    The `DynAny` has components and the current position is -1.

[TypeMismatch](#)    The accessed component in the `DynAny` is of a type that is not equivalent to the inserted type.

### See Also

[DynamicAny::DynAny::get\\_longlong\(\)](#)

## DynAny::insert\_octet()

```
// Java
void insert_octet(
    byte value
);
```

Inserts an `byte` value into the `DynAny` object.

### Parameters

`value`            The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_octet` (an `octet` [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_octet`. The current position is unchanged after the call.

---

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::get\\_octet\(\)](#)

## DynAny::insert\_reference()

```
// Java
void insert_reference(
    org.omg.CORBA.Object value
);
```

Inserts an [Object](#) reference into the `DynAny` object.

## Parameters

`value` The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_Object` (an object reference [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_Object`. The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::get\\_reference\(\)](#)

## DynAny::insert\_short()

```
// Java
void insert_short(
    short value
);
```

Inserts a `short` value into the `DynAny` object.

---

## Parameters

value            The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_short` (a short [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_short`. The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#)    The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#)    The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::get\\_short\(\)](#)

## `DynAny::insert_string()`

```
// Java
void insert_string(
    java.lang.String value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Inserts a string into the `DynAny` object.

## Parameters

value            The value to insert into the `DynAny` object.

You can insert both bounded and unbounded strings using `insert_string()`.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_string` (a string [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_string`. The current position is unchanged after the call.

---

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the inserted type.

[InvalidValue](#)

- The `DynAny` has components and the current position is `-1`.
- The string inserted is longer than the bound of a bounded string.

## See Also

[DynamicAny::DynAny::get\\_string\(\)](#)

## DynAny::insert\_typecode()

```
// Java
void insert_typecode(
    org.omg.CORBA.TypeCode value
);
```

Inserts a [TypeCode](#) value into the `DynAny` object.

## Parameters

value                    The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_TypeCode` (a [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_TypeCode`. The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is `-1`.

## See Also

[DynamicAny::DynAny::get\\_typecode\(\)](#)

## DynAny::insert\_ulong()

```
// Java
void insert_ulong(
```



---

```
        int value
    );
```

Inserts a `int` value into the `DynAny` object.

### Parameters

`value`            The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_ulong` (an unsigned long [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_ulong`. The current position is unchanged after the call.

### Exceptions

[TypeMismatch](#)    The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#)    The `DynAny` has components and the current position is -1.

### See Also

[DynamicAny::DynAny::get\\_ulong\(\)](#)

## **DynAny::insert\_ulonglong()**

```
// Java
void insert_ulonglong(
    long value
);
```

Inserts a `long` value into the `DynAny` object.

### Parameters

`value`            The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_ulonglong` (an unsigned long long [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_ulonglong`. The current position is unchanged after the call.

---

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::get\\_ulonglong\(\)](#)

## DynAny::insert\_ushort()

```
// Java
void insert_ushort(
    short value
);
```

Inserts a `short` value into the `DynAny` object.

## Parameters

`value` The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_ushort` (an unsigned short [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_ushort`. The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#) The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#) The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::get\\_ushort\(\)](#)

## DynAny::insert\_val()

```
// Java
void insert_val(
    java.io.Serializable value
);
```

Inserts a value type value into the `DynAny` object.

---

## Parameters

value            The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to a value type [TypeCode](#), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to a value type [TypeCode](#). The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#)    The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#)    The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::get\\_val\(\)](#)

## `DynAny::insert_wchar()`

```
// Java
void insert_wchar(
    char value
);
```

Inserts a `char` value into the `DynAny` object.

## Parameters

value            The value to insert into the `DynAny` object.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_wchar` (a wide character [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_wchar`. The current position is unchanged after the call.

## Exceptions

[TypeMismatch](#)    The accessed component in the `DynAny` is of a type that is not equivalent to the requested type.

[InvalidValue](#)    The `DynAny` has components and the current position is -1.

## See Also

[DynamicAny::DynAny::get\\_wchar\(\)](#)

---

## DynAny::insert\_wstring()

```
// Java
void insert_wstring(
    java.lang.String value
);
```

Inserts a wide string into the `DynAny` object.

### Parameters

`value`            The value to insert into the `DynAny` object.

You can insert both bounded and unbounded strings using `insert_wstring()`.

It is valid for you to use this method if the [TypeCode](#) contained in the `DynAny` is equivalent to `_tc_wstring` (a wide string [TypeCode](#)), or, if the [TypeCode](#) at the current position (a `DynAny` objects with components) is equivalent to `_tc_wstring`. The current position is unchanged after the call.

### Exceptions

[TypeMismatch](#)    The accessed component in the `DynAny` is of a type that is not equivalent to the inserted type.

[InvalidValue](#)    • The `DynAny` has components and the current position is -1.  
• The string inserted is longer than the bound of a bounded string.

### See Also

[DynamicAny::DynAny::get\\_wstring\(\)](#)

## DynAny::InvalidValue User Exception

```
// Java
package org.omg.DynamicAny.DynAnyPackage;
public final class InvalidValue
    extends org.omg.CORBA.UserException
{
    public InvalidValue() {
        super(InvalidValueHelper.id());
    }
}
```

---

```
}
```

A user exception meaning that an invalid value has been used as a parameter.

**See Also**

[DynamicAny::DynAny::TypeMismatch](#)

## DynAny::next()

```
// Java
boolean next();
```

Advances the current position to the next component of the `DynAny` object. Returns true if the resulting current position indicates a component, false otherwise. Invoking `next()` on a `DynAny` that has no components returns false. A false return value always sets the current position to -1.

**See Also**

[DynamicAny::DynAny::component\\_count\(\)](#)  
[DynamicAny::DynAny::current\\_component\(\)](#)  
[DynamicAny::DynAny::seek\(\)](#)  
[DynamicAny::DynAny::rewind\(\)](#)

## DynAny::rewind()

```
// Java
void rewind();
```

Sets the current position to the first component of the `DynAny` object. This is equivalent to calling `seek(0)`.

**See Also**

[DynamicAny::DynAny::seek\(\)](#)

## DynAny::seek()

```
// Java
boolean seek(
    int index
);
```

Sets the current position to a component of the `DynAny` object. The method returns true if the resulting current position indicates a component of the `DynAny` object and false if the position does not correspond to a component.

---

## Parameters

`index` The new index to set the current position to. An index can range from 0 to  $n-1$ . An index of zero corresponds to the first component.

Calling `seek` with a negative index is legal and sets the current position to -1 to indicate no component. The method returns false in this case.

Passing a non-negative index value for a `DynAny` that does not have a component at the corresponding position sets the current position to - 1 and returns false.

## See Also

[DynamicAny::DynAny::component\\_count\(\)](#)  
[DynamicAny::DynAny::current\\_component\(\)](#)  
[DynamicAny::DynAny::rewind\(\)](#)  
[DynamicAny::DynAny::next\(\)](#)

## DynAny::to\_any()

```
// Java
org.omg.CORBA.Any to_any();
```

Returns an [Any](#) value created from a `DynAny` object. A copy of the [TypeCode](#) associated with the `DynAny` object is assigned to the resulting any. The value associated with the `DynAny` object is copied into the [Any](#) value.

## See Also

[DynamicAny::DynAny::from\\_any\(\)](#)

## DynAny::type()

```
// Java
org.omg.CORBA.TypeCode type();
```

Returns the [TypeCode](#) associated with a `DynAny` object.

A `DynAny` object is created with a [TypeCode](#) value assigned to it. This value determines the type of the value handled through the `DynAny` object. `type()` returns the [TypeCode](#) associated with a `DynAny` object.

---

Note that the [TypeCode](#) associated with a `DynAny` object is initialized at the time the `DynAny` is created and cannot be changed during the lifetime of the `DynAny` object.

## DynAny::TypeMismatch User Exception

```
// Java
package org.omg.DynamicAny.DynAnyPackage;
public final class TypeMismatch
    extends org.omg.CORBA.UserException
{
    public TypeMismatch() {
        super(TypeMismatchHelper.id());
    }
}
```

A user exception meaning that the type of a parameter does not match the type of the target.

This exception is also raised when attempts are made to access `DynAny` components illegally. For example:

- If an attempt is made to access an object's component but the type of object does not have components.
- If an attempt is made to call an `insert_type()` or `get_type()` method on a `DynAny` object whose current component itself has components.

### See Also

[DynamicAny::DynAny::InvalidValue](#)





# DynamicAny::DynAnyFactory Class

You can create [DynAny](#) objects by invoking operations on the `DynAnyFactory` object. You obtain a reference to the `DynAnyFactory` object by calling [CORBA.ORB.resolve\\_initial\\_references\(\)](#) with the identifier parameter set to "DynAnyFactory".

A typical first step in dynamic interpretation of an [Any](#) involves creating a [DynAny](#) object using [create\\_dyn\\_any\(\)](#) or [create\\_dyn\\_any\\_from\\_type\\_code\(\)](#). Then, depending on the type of the [Any](#), you narrow the resulting [DynAny](#) object reference to one of the following complex types of object references:

[DynFixed](#)  
[DynStruct](#)  
[DynSequence](#)  
[DynArray](#)  
[DynUnion](#)  
[DynEnum](#)  
[DynValue](#)

Finally, you can use [DynAny.to\\_any\(\)](#) (which each of these classes inherits from the [DynAny](#) class) to create an [Any](#) value from the constructed [DynAny](#).

## Exceptions

**MARSHAL:** an attempt is made to exported references to `DynAnyFactory` objects to other processes or if an attempt is made to externalized them with [ORB.object\\_to\\_string\(\)](#). `DynAnyFactory` objects are intended to be local to the process in which they are created and used.

```
package org.omg.DynamicAny;

public interface DynAnyFactory extends org.omg.CORBA.Object
{
    org.omg.DynamicAny.DynAny create_dyn_any(
        org.omg.CORBA.Any value
    ) throws
        org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode;

    org.omg.DynamicAny.DynAny create_dyn_any_from_type_code(
        org.omg.CORBA.TypeCode type
    ) throws
        org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode;
}
```

---

```
        ) throws
        org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode;
    }
```

## DynAnyFactory::create\_dyn\_any()

```
// Java
org.omg.DynamicAny.DynAny create_dyn_any(
    org.omg.CORBA.Any value
);
```

Returns a new [DynAny](#) object from an [Any](#) value.

### Parameters

value            An [Any](#) value to use to set the [DynAny](#) object.

A copy of the [TypeCode](#) associated with the any value is assigned to the resulting [DynAny](#) object. The value associated with the [DynAny](#) object is a copy of the value in the original [Any](#). The current position of the created [DynAny](#) object is set to zero if the passed value has components; otherwise, the current position is set to -1.

### Exceptions

[InconsistentTypeCode](#): the value has a [TypeCode](#) with a [TCKind](#) of tk\_Principal, tk\_native, Or tk\_abstract\_interface.

### See Also

[DynamicAny.DynAnyFactory.create\\_dyn\\_any\\_from\\_type\\_code\(\)](#)

## DynAnyFactory::create\_dyn\_any\_from\_type\_code()

```
// Java
org.omg.DynamicAny.DynAny create_dyn_any_from_type_code(
    org.omg.CORBA.TypeCode type
);
```

Returns a new [DynAny](#) object from a [TypeCode](#) value. Depending on the [TypeCode](#), the created object may be of type [DynAny](#), or one of its derived types, such as [DynStruct](#). The returned reference can be narrowed to the derived type.

### Parameters

type            A [TypeCode](#) value to use to set the [DynAny](#) object.

---

Table 13 shows the initial default values set depending on the type created:

**Table 13:** *Default Values When Using create\_dyn\_any\_from\_type\_code()*

Type	Default Value
<a href="#">Any</a> values	An <a href="#">Any</a> containing a <a href="#">TypeCode</a> with a <a href="#">TCKind</a> value of <code>tk_null</code> and no value.
boolean	FALSE
char	zero
<a href="#">DynArray</a>	The operation sets the current position to zero and recursively initializes elements to their default value.
<a href="#">DynEnum</a>	The operation sets the current position to -1 and sets the value of the enumerator to the first enumerator value indicated by the <a href="#">TypeCode</a> .
<a href="#">DynFixed</a>	Operations set the current position to -1 and sets the value to zero.
<a href="#">DynSequence</a>	The operation sets the current position to -1 and creates an empty sequence.
<a href="#">DynStruct</a>	The operation sets the current position to -1 for empty exceptions and to zero for all other <a href="#">TypeCode</a> values. The members (if any) are recursively initialized to their default values.
<a href="#">DynUnion</a>	The operation sets the current position to zero. The discriminator value is set to a value consistent with the first named member of the union. That member is activated and recursively initialized to its default value.
<a href="#">DynValue</a>	The members are initialized as for a <a href="#">DynStruct</a> .
numeric types	zero
object references	nil

---

---

**Table 13:** *Default Values When Using create\_dyn\_any\_from\_type\_code()*

Type	Default Value
octet	zero
string	the empty string
<a href="#">TypeCode</a>	A <a href="#">TypeCode</a> with a <a href="#">TCKind</a> value of tk_null
wchar	zero
wstring	the empty string

**Exceptions**

[InconsistentTypeCode](#): the [TypeCode](#) has a [TCKind](#) of tk\_Principal, tk\_native, or tk\_abstract\_interface.

**See Also**

[DynamicAny.DynAnyFactory.create\\_dyn\\_any\(\)](#)

### DynAnyFactory::InconsistentTypeCode User Exception Class

```
// Java
package org.omg.DynamicAny.DynAnyFactoryPackage;
public final class InconsistentTypeCode
    extends org.omg.CORBA.UserException
{
    public InconsistentTypeCode() {
        super(InconsistentTypeCodeHelper.id());
    }
}
```

A user exception meaning that a parameter has an inconsistent [TypeCode](#) compared to the object.

# DynamicAny::DynArray Class

DynArray objects let you dynamically manipulate [Any](#) values as arrays. The following methods let you get and set array elements:

[get\\_elements\(\)](#)  
[set\\_elements\(\)](#)  
[get\\_elements\\_as\\_dyn\\_any\(\)](#)  
[set\\_elements\\_as\\_dyn\\_any\(\)](#)

This class inherits from the [DynAny](#) class. Use [component\\_count\(\)](#) to get the dimension of the array. Use the iteration methods such as [seek\(\)](#) to access portions of the array.

```
// Java
package org.omg.DynamicAny;

public interface DynArray extends
    org.omg.DynamicAny.DynAny
{
    org.omg.CORBA.Any[] get_elements();

    void set_elements(
        org.omg.CORBA.Any[] value
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
        org.omg.DynamicAny.DynAnyPackage.InvalidValue;

    org.omg.DynamicAny.DynAny[] get_elements_as_dyn_any();

    void set_elements_as_dyn_any(
        org.omg.DynamicAny.DynAny[] value
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
        org.omg.DynamicAny.DynAnyPackage.InvalidValue;
}
```

**See Also**

[DynamicAny::DynAny](#)

---

## DynArray::get\_elements()

```
// Java
org.omg.CORBA.Any[] get_elements();
```

Returns a sequence of [Any](#) values containing the elements of the array.

### See Also

[DynamicAny::DynArray::set\\_elements\(\)](#)  
[DynamicAny::DynArray::get\\_elements\\_as\\_dyn\\_any\(\)](#)  
[DynamicAny::DynAny::component\\_count\(\)](#)

## DynArray::get\_elements\_as\_dyn\_any()

```
// Java
org.omg.DynamicAny.DynAny[] get_elements_as_dyn_any();
```

Returns a sequence of [DynAny](#) objects that describes each member in the array.

Use this method instead of [get\\_elements\(\)](#) if you want to avoid converting [DynAny](#) objects to [Any](#) objects when your application needs to handle [DynArray](#) objects extensively.

### See Also

[DynamicAny::DynArray::get\\_elements\(\)](#)  
[DynamicAny::DynArray::set\\_elements\\_as\\_dyn\\_any\(\)](#)  
[DynamicAny::DynAny::component\\_count\(\)](#)

## DynArray::set\_elements()

```
// Java
void set_elements(
    org.omg.CORBA.Any[] value
) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Sets the array values with a sequence of [Any](#) values.

### Parameters

value            A sequence of [Any](#) values containing the elements for the array.

This method sets the current position to -1 if the sequence has a zero length and it sets it to 0 otherwise.

---

**Exceptions**     [TypeMismatch](#) is raised if an inconsistent value is passed in the sequence.  
[InvalidValue](#) is raised if the sequence length does not match the array length.

**See Also**     [DynamicAny::DynArray::get\\_elements\(\)](#)  
[DynamicAny::DynArray::set\\_elements\\_as\\_dyn\\_any\(\)](#)  
[DynamicAny::DynAny::component\\_count\(\)](#)

## DynArray::set\_elements\_as\_dyn\_any()

```
// Java
void set_elements_as_dyn_any(
    org.omg.DynamicAny.DynAny[] value
) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Initializes the array data associated with a `DynArray` object from a sequence of [DynAny](#) objects. Use this method instead of [set\\_elements\(\)](#) if you want to avoid converting [DynAny](#) objects to [Any](#) objects when your application needs to handle `DynArray` objects extensively.

### Parameters

value             A sequence of [DynAny](#) objects representing the array elements.

This method sets the current position to -1 if the sequence has a zero length and it sets it to 0 otherwise.

**Exceptions**     [TypeMismatch](#) is raised if an inconsistent value is passed in the sequence.  
[InvalidValue](#) is raised if the sequence length does not match the array length.

**See Also**     [DynamicAny::DynArray::get\\_elements\\_as\\_dyn\\_any\(\)](#)  
[DynamicAny::DynArray::set\\_elements\(\)](#)  
[DynamicAny::DynAny::component\\_count\(\)](#)





# DynamicAny::DynEnum Class

A `DynEnum` object lets you dynamically manipulate an [Any](#) value as an enumerated value. The key methods allow you to get and set a value as an IDL identifier string or you can manipulate the number that the enumerated value represents:

[get\\_as\\_string\(\)](#)  
[set\\_as\\_string\(\)](#)  
[get\\_as\\_ulong\(\)](#)  
[set\\_as\\_ulong\(\)](#)

This class inherits from the [DynAny](#) class. The current position of a `DynEnum` is always -1 because it can only be one value at a given time.

```
// Java
package org.omg.DynamicAny;

public interface DynEnum extends
    org.omg.DynamicAny.DynAny
{
    java.lang.String value_as_string();

    void set_value_as_string(
        java.lang.String value
    ) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

    int get_as_ulong();

    void set_as_ulong(
        int value
    ) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;
}
```

**See Also**

[DynamicAny::DynAny](#)

---

## DynEnum::value\_as\_string()

```
// Java
java.lang.String value_as_string();
```

Returns a string for the `DynEnum` that represents the IDL enumeration identifier.

### See Also

[DynamicAny::DynEnum::set\\_as\\_string\(\)](#)  
[DynamicAny::DynEnum::get\\_as\\_ulong\(\)](#)

## DynEnum::get\_as\_ulong()

```
// Java
int get_as_ulong();
```

Returns a number for the `DynEnum` that represents the enumerated ordinal value. Enumerators have ordinal values of 0 to  $n-1$ , as they appear from left to right in the corresponding IDL definition.

### See Also

[DynamicAny::DynEnum::set\\_as\\_ulong\(\)](#)  
[DynamicAny::DynEnum::value\\_as\\_string\(\)](#)

## DynEnum::set\_as\_string()

```
// Java
void set_value_as_string(
    java.lang.String value
) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Sets the enumerated identifier string value for the `DynEnum`.

### Parameters

`value`            The identifier string to set the enumerated value to.

### Exceptions

[InvalidValue](#)    The `value` string is not a valid IDL identifier for the corresponding IDL enumerated type.

### See Also

[DynamicAny::DynEnum::value\\_as\\_string\(\)](#)  
[DynamicAny::DynEnum::set\\_as\\_ulong\(\)](#)

---

## DynEnum::set\_as\_ulong()

Sets the numerical value for the `DynEnum` that represents the enumerated ordinal value.

### Parameters

`value`            The number to set the enumerated value to.

### Exceptions

[InvalidValue](#)    The `value` is outside the range of ordinal values for the corresponding IDL enumerated type.

### See Also

[DynamicAny::DynEnum::get\\_as\\_ulong\(\)](#)  
[DynamicAny::DynEnum::set\\_as\\_string\(\)](#)



# DynamicAny::DynFixed Class

A `DynFixed` object lets you dynamically manipulate an [Any](#) value as a fixed point value. This class inherits from the [DynAny](#) class. The key methods include [get\\_value\(\)](#) and [set\\_value\(\)](#).

These methods use strings to represent fixed-point values. A fixed-point format consists of an integer part of digits, a decimal point, a fraction part of digits, and a `d` or `D`. Examples include:

```
1.2d
35.98D
456.32
.467
```

Either the integer part or the fraction part (but not both) may be missing. The decimal point is not required for whole numbers. The `d` or `D` are optional. leading or trailing white space is allowed.

```
// Java
package org.omg.DynamicAny;

public interface DynFixed extends
    org.omg.DynamicAny.DynAny
{
    java.lang.String get_value();

    boolean set_value(
        java.lang.String val
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
        org.omg.DynamicAny.DynAnyPackage.InvalidValue;
}
```

**See Also**

[DynamicAny::DynAny](#)

---

## DynFixed::get\_value()

```
// Java
java.lang.String get_value();
```

Returns a string representing the fixed value of the `DynFixed` object.

**See Also** [DynamicAny::DynFixed::set\\_value\(\)](#)

## DynFixed::set\_value()

```
// Java
boolean set_value(
    java.lang.String val
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Sets the value of the `DynFixed`. The method returns true if `val` can be represented as the `DynFixed` without loss of precision. If `val` has more fractional digits than can be represented in the `DynFixed`, the fractional digits are truncated and the method returns false.

### Parameters

`val` A string containing the fixed point value to be set in the `DynFixed`. The string must contain a fixed string constant in the same format as would be used for IDL fixed-point literals. However, the trailing `d` or `D` is optional.

### Exceptions

[InvalidValue](#) `val` contains a value whose scale exceeds that of the `DynFixed` or is not initialized.

[TypeMismatch](#) `val` does not contain a valid fixed-point literal or contains extraneous characters other than leading or trailing white space.

**See Also** [DynamicAny::DynFixed::get\\_value\(\)](#)

# DynamicAny::DynSequence Class

DynSequence objects let you dynamically manipulate [Any](#) values as sequences. The key methods allow you to manage the sequence length and get and set sequence elements:

[get\\_length\(\)](#)  
[set\\_length\(\)](#)  
[get\\_elements\(\)](#)  
[set\\_elements\(\)](#)  
[get\\_elements\\_as\\_dyn\\_any\(\)](#)  
[set\\_elements\\_as\\_dyn\\_any\(\)](#)

This class inherits from the [DynAny](#) class.

```
// Java
package org.omg.DynamicAny;

public interface DynSequence extends
    org.omg.DynamicAny.DynAny
{
    int length();

    void set_length(
        int len
    ) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

    org.omg.CORBA.Any[] get_elements();

    void set_elements(
        org.omg.CORBA.Any[] value
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
        org.omg.DynamicAny.DynAnyPackage.InvalidValue;

    org.omg.DynamicAny.DynAny[] get_elements_as_dyn_any();

    void set_elements_as_dyn_any(
        org.omg.DynamicAny.DynAny[] value
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
        org.omg.DynamicAny.DynAnyPackage.InvalidValue;
}
```

---

```
}
```

**See Also**

[DynamicAny](#): [DynAny](#)

## **DynSequence::get\_elements()**

```
org.omg.CORBA.Any[] get_elements();
```

Returns a sequence of [Any](#) values containing the elements of the sequence.

**See Also**

[DynamicAny](#): [DynSequence](#): [set\\_elements\(\)](#)  
[DynamicAny](#): [DynSequence](#): [get\\_elements\\_as\\_dyn\\_any\(\)](#)

## **DynSequence::get\_elements\_as\_dyn\_any()**

```
org.omg.DynamicAny.DynAny[] get_elements_as_dyn_any();
```

Returns a sequence of [DynAny](#) objects that describes each member in the sequence.

Use this method instead of [get\\_elements\(\)](#) if you want to avoid converting [DynAny](#) objects to [Any](#) objects when your application needs to handle [DynSequence](#) objects extensively.

**See Also**

[DynamicAny](#): [DynSequence](#): [get\\_elements\(\)](#)  
[DynamicAny](#): [DynSequence](#): [get\\_elements\\_as\\_dyn\\_any\(\)](#)

## **DynSequence::length()**

```
int length();
```

Returns the number of elements in the sequence.

**See Also**

[DynamicAny](#): [DynSequence](#): [set\\_length\(\)](#)  
[DynamicAny](#): [DynSequence](#): [get\\_elements\(\)](#)



---

## DynSequence::set\_elements()

```
void set_elements(  
    org.omg.CORBA.Any[] value  
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,  
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Sets the sequence values.

### Parameters

value            A sequence of [Any](#) values containing the elements for the sequence.

This method sets the current position to -1 if the sequence has a zero length and it sets it to 0 otherwise.

### Exceptions

[Invalidvalue](#)    The parameter's length is greater than the `DynSequence` length.

[TypeMismatch](#)    an inconsistent value is passed in. This can happen if:

- The element type codes between the `DynSequence` and the parameter do not agree.
- The `DynSequence` is a bounded sequence and the number of elements in the parameter are greater than the bound allows.

### See Also

[DynamicAny::DynSequence::get\\_elements\(\)](#)  
[DynamicAny::DynSequence::set\\_elements\\_as\\_dyn\\_any\(\)](#)  
[DynamicAny::DynSequence::length\(\)](#)  
[DynamicAny::DynSequence::set\\_length\(\)](#)

## DynSequence::set\_elements\_as\_dyn\_any()

```
void set_elements_as_dyn_any(  
    org.omg.DynamicAny.DynAny[] value  
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,  
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Initializes the sequence data associated with a `DynSequence` object from a sequence of [DynAny](#) objects. Use this method instead of [set\\_elements\(\)](#) if you

---

want to avoid converting [DynAny](#) objects to [Any](#) objects when your application needs to handle `DynSequence` objects extensively.

## Parameters

`value`            A sequence of [DynAny](#) objects to represent the elements of the `DynSequence`.

This method sets the current position to -1 if the sequence has a zero length and it sets it to 0 otherwise.

## Exceptions

`Invalidvalue`    The parameter's length is greater than the `DynSequence` length.

[TypeMismatch](#)    An inconsistent value is passed in. This can happen if:

- The element type codes between the `DynSequence` and the parameter do not agree.
- The `DynSequence` is a bounded sequence and the number of elements in the parameter are greater than the bound allows.

## See Also

[DynamicAny::DynSequence::get\\_elements\\_as\\_dyn\\_any\(\)](#)  
[DynamicAny::DynSequence::set\\_elements\(\)](#)  
[DynamicAny::DynSequence::length\(\)](#)  
[DynamicAny::DynSequence::set\\_length\(\)](#)

## `DynSequence::set_length()`

Sets the length of the sequence.

## Parameters

`len`                The length desired for the sequence.

Increasing the length adds new (default-initialized) elements to the end of the sequence without affecting existing elements in the sequence. The new current position is set to the first new element if the previous current position was -1. The new current position remains the same as the old one if the previous current position indicates a valid element (was anything but -1).

---

Decreasing the length removes elements from the end of the sequence without affecting the rest of the elements. The new current position is as follows:

- If the previous current position indicates a valid element and that element is not removed, the new current position remains the same.
- If the previous current position indicates a valid element and that element is removed, the new current position is set to -1.
- If the sequence length is set to 0, the new current position is set to -1.
- If the previous current position was -1, the new current position remains -1.

### Exceptions

[InvalidValue](#) An attempt is made to increase the length of a bounded sequence to a value greater than the bound.

### See Also

[DynamicAny](#): [DynSequence](#): [length\(\)](#)  
[DynamicAny](#): [DynSequence](#): [set\\_elements\(\)](#)



# DynamicAny::DynStruct Class

You use `DynStruct` objects for dynamically handling structures and exceptions in [Any](#) values. This class inherits from the [DynAny](#) class. Key methods allow you to set and get the structure (or exception) as a sequence of name-value pairs:

[get\\_members\(\)](#)  
[set\\_members\(\)](#)  
[get\\_members\\_as\\_dyn\\_any\(\)](#)  
[set\\_members\\_as\\_dyn\\_any\(\)](#)

Use the [DynAny](#) iteration methods such as [seek\(\)](#) to set the current position to a member of the structure. You can also obtain the name and kind of [TypeCode](#) for a member at the current position:

[current\\_member\\_name\(\)](#)  
[current\\_member\\_kind\(\)](#)

```
// Java
package org.omg.DynamicAny;

public interface DynStruct extends
    org.omg.DynamicAny.DynAny
{
    java.lang.String current_member_name() throws
    org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    org.omg.CORBA.TCKind current_member_kind() throws
    org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    org.omg.DynamicAny.NameValuePair[] get_members();

    void set_members(
        org.omg.DynamicAny.NameValuePair[] value
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;

    org.omg.DynamicAny.NameDynAnyPair[] get_members_as_dyn_any();
```

---

```
void set_members_as_dyn_any(  
    org.omg.DynamicAny.NameDynAnyPair[] value  
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,  
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;  
  
}
```

**See Also** [DynamicAny::DynAny](#)

### **DynStruct::current\_member\_kind()**

```
org.omg.CORBA.TCKind current_member_kind() throws  
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;
```

Returns the kind of [TypeCode](#) associated with the current position.

#### **Exceptions**

[TypeMismatch](#) The `DynStruct` object represents an empty exception.

[InvalidValue](#) The current position does not indicate a member.

**See Also** [DynamicAny::DynAny::seek\(\)](#)  
[DynamicAny::DynStruct::current\\_member\\_name\(\)](#)

### **DynStruct::current\_member\_name()**

```
java.lang.String current_member_name() throws  
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;
```

Returns the name of the member at the current position. This method can return an empty value since the [TypeCode](#) of the value being manipulated may not contain the names of members.

#### **Exceptions**

[TypeMismatch](#) `DynStruct` object represents an empty exception.

[InvalidValue](#) The current position does not indicate a member.

**See Also** [DynamicAny::DynAny::seek\(\)](#)  
[DynamicAny::DynStruct::current\\_member\\_kind\(\)](#)

---

## DynStruct::get\_members()

```
org.omg.DynamicAny.NameValuePair[] get_members();
```

Returns a sequence of members that describes the name and the value of each member in the structure (or exception) associated with a `DynStruct` object.

The sequence order is the same as the declaration order of members as indicated by the [TypeCode](#) of the `DynStruct`. The current position is not affected. The member names in the returned sequence will be empty strings if the [TypeCode](#) of the `DynStruct` does not contain member names.

### See Also

[DynamicAny::DynStruct::set\\_members\(\)](#)  
[DynamicAny::DynStruct::get\\_members\\_as\\_dyn\\_any\(\)](#)

## DynStruct::get\_members\_as\_dyn\_any()

```
org.omg.DynamicAny.NameDynAnyPair[] get_members_as_dyn_any();
```

Returns a sequence of name-[DynAny](#) pairs that describes each member in the structure (or exception) associated with a `DynStruct` object. Use this method instead of [get\\_members\(\)](#) if you want to avoid converting [DynAny](#) objects to any objects when your application needs to handle `DynStruct` objects extensively.

The sequence order is the same as the declaration order of members as indicated by the [TypeCode](#) of the `DynStruct`. The current position is not affected. The member names in the returned sequence will be empty strings if the [TypeCode](#) of the `DynStruct` does not contain member names.

### See Also

[DynamicAny::DynStruct::set\\_members\\_as\\_dyn\\_any\(\)](#)  
[DynamicAny::DynStruct::get\\_members\(\)](#)

## DynStruct::set\_members()

```
void set_members(  
    org.omg.DynamicAny.NameValuePair[] value  
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,  
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Initializes the structure data associated with a `DynStruct` object from a sequence of name-value pairs.

---

## Parameters

**value** A sequence of name-value pairs representing member names and the values of the members.

The current position is set to zero if the sequence passed in has a non-zero length. The current position is set to -1 if an empty sequence is passed in.

Members in the sequence must follow these rules:

- Members must be in the order in which they appear in the IDL specification of the structure.
- If member names are supplied in the sequence, they must either match the corresponding member name in the [TypeCode](#) of the `DynStruct` or they must be empty strings.
- Members must be supplied in the same order as indicated by the [TypeCode](#) of the `DynStruct`. The method does not reassign member values based on member names.

## Exceptions

[InvalidValue](#) The sequence has a number of elements that disagrees with the number of members as indicated by the [TypeCode](#) of the `DynStruct`.

[TypeMismatch](#) Raised if:

- One or more sequence elements have a type that is not equivalent to the [TypeCode](#) of the corresponding member.
- The member names do not match the corresponding member name in the [TypeCode](#) of the `DynStruct`.

## See Also

[DynamicAny::DynStruct::get\\_members\(\)](#)  
[DynamicAny::DynStruct::set\\_members\\_as\\_dyn\\_any\(\)](#)

## DynStruct::set\_members\_as\_dyn\_any()

```
void set_members_as_dyn_any(  
    org.omg.DynamicAny.NameDynAnyPair[] value  
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,  
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```



---

Initializes the structure data associated with a `DynStruct` object from a sequence of name-`DynAny` pairs. Use this method instead of `set_members()` if you want to avoid converting `DynAny` objects to `any` objects when your application needs to handle `DynStruct` objects extensively.

## Parameters

`value`                    A sequence of name-`DynAny` pairs representing member names and the values of the members as `DynAny` objects.

The current position is set to zero if the sequence passed in has a non-zero length. The current position is set to -1 if an empty sequence is passed in.

Members in the sequence must follow these rules:

- Members must be in the order in which they appear in the IDL specification of the structure.
- If member names are supplied in the sequence, they must either match the corresponding member name in the `TypeCode` of the `DynStruct` or they must be empty strings.
- Members must be supplied in the same order as indicated by the `TypeCode` of the `DynStruct`. The method does not reassign `DynAny` values based on member names.

## Exceptions

[`InvalidValue`](#)    The sequence has a number of elements that disagrees with the number of members as indicated by the `TypeCode` of the `DynStruct`.

[`TypeMismatch`](#)    Raised if:

- One or more sequence elements have a type that is not equivalent to the `TypeCode` of the corresponding member.
- The member names do not match the corresponding member name in the `TypeCode` of the `DynStruct`.

## See Also

[`DynamicAny::DynStruct::get\_members\_as\_dyn\_any\(\)`](#)  
[`DynamicAny::DynStruct::set\_members\(\)`](#)



# DynamicAny::DynUnion Class

The `DynUnion` class lets you dynamically manage an [Any](#) value as a union value. This class inherits from the [DynAny](#) class. Key methods to manipulate a union include:

[has\\_no\\_active\\_member\(\)](#)  
[member\(\)](#)  
[member\\_kind\(\)](#)  
[member\\_name\(\)](#)

Other methods are available to manipulate a union's discriminator:

[discriminator\\_kind\(\)](#)  
[discriminator\(\)](#)  
[set\\_discriminator\(\)](#)  
[set\\_to\\_default\\_member\(\)](#)  
[set\\_to\\_no\\_active\\_member\(\)](#)

A union can have only two valid current positions: Zero denotes the discriminator and 1 denotes the active member.

The value returned by `DynAny::component_count()` for a union depends on the current discriminator: it is 2 for a union whose discriminator indicates a named member, and 1 otherwise.

```
// Java
package org.omg.DynamicAny;

public interface DynUnion extends
    org.omg.DynamicAny.DynAny
{
    org.omg.DynamicAny.DynAny discriminator();

    void set_discriminator(
        org.omg.DynamicAny.DynAny d
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

    void set_to_default_member() throws
        org.omg.DynamicAny.DynAnyPackage.TypeMismatch;
}
```

---

```
void set_to_no_active_member() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

boolean has_no_active_member();

org.omg.CORBA.TCKind discriminator_kind();

org.omg.DynamicAny.DynAny member() throws
org.omg.DynamicAny.DynAnyPackage.InvalidValue;

java.lang.String member_name() throws
org.omg.DynamicAny.DynAnyPackage.InvalidValue;

org.omg.CORBA.TCKind member_kind() throws
org.omg.DynamicAny.DynAnyPackage.InvalidValue;

}
```

**See Also** [DynamicAny::DynAny](#)

### **DynUnion::discriminator\_kind()**

```
org.omg.CORBA.TCKind discriminator_kind();
```

Returns the kind of [TypeCode](#) of the union's discriminator.

**See Also** [DynamicAny::DynUnion::discriminator\(\)](#)  
[DynamicAny::DynUnion::set\\_discriminator\(\)](#)

### **DynUnion::discriminator()**

```
org.omg.DynamicAny.DynAny discriminator();
```

Returns the current discriminator value of the `DynUnion`.

**See Also** [DynamicAny::DynUnion::set\\_discriminator\(\)](#)  
[DynamicAny::DynUnion::discriminator\\_kind\(\)](#)

---

## DynUnion::has\_no\_active\_member()

```
boolean has_no_active_member();
```

Returns true if the union has no active member (that is, the union's value consists solely of its discriminator because the discriminator has a value that is not listed as an explicit case label). The method returns false if:

- The IDL union has a default case.
- The IDL union's explicit case labels use the entire range of discriminator values.

### See Also

[DynamicAny::DynUnion::member\(\)](#)  
[DynamicAny::DynUnion::set\\_to\\_default\\_member\(\)](#)  
[DynamicAny::DynUnion::set\\_to\\_no\\_active\\_member\(\)](#)

## DynUnion::member()

```
org.omg.DynamicAny.DynAny member() throws  
org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Returns the currently active member. Note that the returned reference remains valid only for as long as the currently active member does not change.

### Parameters

[InvalidValue](#) The union has no active member.

### See Also

[DynamicAny::DynUnion::member\\_kind\(\)](#)  
[DynamicAny::DynUnion::member\\_name\(\)](#)  
[DynamicAny::DynUnion::has\\_no\\_active\\_member\(\)](#)

## DynUnion::member\_kind()

```
org.omg.CORBA.TCKind member_kind() throws  
org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Returns the kind of [TypeCode](#) of the currently active member.

### Exceptions

[InvalidValue](#) The method is called on a union without an active member.

---

**See Also**

[DynamicAny::DynUnion::member\(\)](#)  
[DynamicAny::DynUnion::member\\_name\(\)](#)

## DynUnion::member\_name()

```
java.lang.String member_name() throws  
org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Returns the name of the currently active member. The method returns an empty string if the union's [TypeCode](#) does not contain a member name for the currently active member.

**Exceptions**

[InvalidValue](#) The method is called on a union without an active member.

**See Also**

[DynamicAny::DynUnion::member\(\)](#)  
[DynamicAny::DynUnion::member\\_kind\(\)](#)

## DynUnion::set\_discriminator()

```
void set_discriminator(  
    org.omg.DynamicAny.DynAny d  
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch;
```

Sets the discriminator of the `DynUnion`.

**Parameters**

`d` The value to set the discriminator to. Setting the discriminator to a value that is consistent with the currently active union member does not affect the currently active member. Setting the discriminator to a value that is inconsistent with the currently active member deactivates the member and activates the member that is consistent with the new discriminator value (if there is a member for that value) by initializing the member to its default value.

Setting the discriminator of a union sets the current position to 0 if the discriminator value indicates a non-existent union member (The method [has\\_no\\_active\\_member\(\)](#) would return true in this case). Otherwise, if the

---

discriminator value indicates a named union member, the current position is set to 1, [has\\_no\\_active\\_member\(\)](#) would return false, and [component\\_count\(\)](#) would return 2 in this case.

### Exceptions

[TypeMismatch](#) The [TypeCode](#) of the parameter is not equivalent to the [TypeCode](#) of the union's discriminator.

### See Also

[DynamicAny::DynUnion::discriminator\(\)](#)  
[DynamicAny::DynUnion::has\\_no\\_active\\_member\(\)](#)  
[DynamicAny::DynUnion::set\\_to\\_default\\_member\(\)](#)  
[DynamicAny::DynUnion::set\\_to\\_no\\_active\\_member\(\)](#)

## DynUnion::set\_to\_default\_member()

void set\_to\_default\_member() throws  
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

Sets the discriminator to a value that is consistent with the value of the default case of a union.

This method sets the current position to zero and causes [component\\_count\(\)](#) to return 2.

### Exceptions

[TypeMismatch](#) The method is called on a union without an explicit default case.

### See Also

[DynamicAny::DynUnion::has\\_no\\_active\\_member\(\)](#)  
[DynamicAny::DynUnion::set\\_discriminator\(\)](#)  
[DynamicAny::DynUnion::set\\_to\\_no\\_active\\_member\(\)](#)  
[DynamicAny::DynUnion::set\\_to\\_no\\_active\\_member\(\)](#)

## DynUnion::set\_to\_no\_active\_member()

void set\_to\_no\_active\_member() throws  
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

Sets the discriminator to a value that does not correspond to any of the union's case labels.

---

This method sets the current position to zero and causes [DynAny::component\\_count\(\)](#) to return 1.

### Exceptions

[TypeMismatch](#) Raised if this method is called on a union that:

- Does not have an explicit default case.
- Uses the entire range of discriminator values for explicit case labels.

### See Also

[DynamicAny::DynUnion::has\\_no\\_active\\_member\(\)](#)  
[DynamicAny::DynUnion::set\\_discriminator\(\)](#)  
[DynamicAny::DynUnion::set\\_to\\_default\\_member\(\)](#)



# DynamicAny::DynValue Class

You use `DynValue` objects for dynamically handling value types in [Any](#) values. Value types are used for objects-by-value. This class inherits from the [DynAny](#) class. Key methods allow you to set and get the value type as a sequence of name-value pairs:

[get\\_members\(\)](#)  
[set\\_members\(\)](#)  
[get\\_members\\_as\\_dyn\\_any\(\)](#)  
[set\\_members\\_as\\_dyn\\_any\(\)](#)

Use the [DynAny](#) iteration methods such as [seek\(\)](#) to set the current position to a member of the value type. You can also obtain the name and kind of [TypeCode](#) for a member at the current position:

[current\\_member\\_name\(\)](#)  
[current\\_member\\_kind\(\)](#)

The class is as follows:

```
// Java
package org.omg.DynamicAny;

public interface DynValue extends
    org.omg.DynamicAny.DynAny
{
    java.lang.String current_member_name() throws
        org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
        org.omg.DynamicAny.DynAnyPackage.InvalidValue;

    org.omg.CORBA.TCKind current_member_kind() throws
        org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
        org.omg.DynamicAny.DynAnyPackage.InvalidValue;

    org.omg.DynamicAny.NameValuePair[] get_members();

    void set_members(
        org.omg.DynamicAny.NameValuePair[] value
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
        org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

---

```
org.omg.DynamicAny.NameDynAnyPair[] get_members_as_dyn_any();

void set_members_as_dyn_any(
    org.omg.DynamicAny.NameDynAnyPair[] value
) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
org.omg.DynamicAny.DynAnyPackage.InvalidValue;

}
```

**See Also**

[DynamicAny::DynAny](#)

### **DynValue::current\_member\_kind()**

```
org.omg.CORBA.TCKind current_member_kind() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Returns the kind of [TypeCode](#) associated with the current position.

**Exceptions**

[TypeMismatch](#) The DynValue object represents an empty value type.

[InvalidValue](#) The current position does not indicate a member.

**See Also**

[DynamicAny::DynAny::seek\(\)](#)

[DynamicAny::DynValue::current\\_member\\_name\(\)](#)

### **DynValue::current\_member\_name()**

```
java.lang.String current_member_name() throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Returns the name of the member at the current position. This method can return an empty value since the [TypeCode](#) of the value being manipulated may not contain the names of members.

---

## Exceptions

[TypeMismatch](#) The `DynValue` object represents an empty value type.

[InvalidValue](#) The current position does not indicate a member.

## See Also

[DynamicAny::DynAny::seek\(\)](#)

[DynamicAny::DynValue::current\\_member\\_kind\(\)](#)

## DynValue::get\_members()

```
org.omg.DynamicAny.NameValuePair[] get_members();
```

Returns a sequence of members that describes the name and the value of each member in the `DynValue` object.

The sequence order is the same as the declaration order of members as indicated by the [TypeCode](#) of the `DynValue`. The current position is not affected. The member names in the returned sequence will be empty strings if the [TypeCode](#) of the `DynValue` does not contain member names.

## See Also

[DynamicAny::DynValue::set\\_members\(\)](#)

[DynamicAny::DynValue::get\\_members\\_as\\_dyn\\_any\(\)](#)

## DynValue::get\_members\_as\_dyn\_any()

```
org.omg.DynamicAny.NameDynAnyPair[] get_members_as_dyn_any();
```

Returns a sequence of name-`DynAny` pairs that describes each member in the value type associated with a `DynValue` object. Use this method instead of [get\\_members\(\)](#) if you want to avoid converting [DynAny](#) objects to [Any](#) objects when your application needs to handle `DynValue` objects extensively.

The sequence order is the same as the declaration order of members as indicated by the [TypeCode](#) of the `DynValue`. The current position is not affected. The member names in the returned sequence will be empty strings if the [TypeCode](#) of the `DynValue` does not contain member names.

## See Also

[DynamicAny::DynValue::set\\_members\\_as\\_dyn\\_any\(\)](#)

[DynamicAny::DynValue::get\\_members\(\)](#)

---

## DynValue::set\_members()

```
void set_members(  
    org.omg.DynamicAny.NameValuePair[] value  
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,  
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Initializes the data value associated with a `DynValue` object from a sequence of name-value pairs.

### Parameters

`values`            A sequence of name-value pairs representing member names and the values of the members.

The current position is set to zero if the sequence passed in has a non-zero length. The current position is set to -1 if an empty sequence is passed in.

Members in the sequence must follow these rules:

- Members must be in the order in which they appear in the IDL specification.
- If member names are supplied in the sequence, they must either match the corresponding member name in the [TypeCode](#) of the `DynValue` or they must be empty strings.
- Members must be supplied in the same order as indicated by the [TypeCode](#) of the `DynValue`. The method does not reassign member values based on member names.

### Exceptions

[InvalidValue](#)    The sequence has a number of elements that disagrees with the number of members as indicated by the [TypeCode](#) of the `DynValue`.

[TypeMismatch](#)    Raised if:

- One or more sequence elements have a type that is not equivalent to the [TypeCode](#) of the corresponding member.
- The member names do not match the corresponding member name in the [TypeCode](#) of the `DynValue`.

### See Also

[DynamicAny::DynValue::get\\_members\(\)](#)  
[DynamicAny::DynValue::set\\_members\\_as\\_dyn\\_any\(\)](#)  
[DynamicAny::NameValuePair](#)

---

## DynValue::set\_members\_as\_dyn\_any()

```
void set_members_as_dyn_any(  
    org.omg.DynamicAny.NameDynAnyPair[] value  
    ) throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch,  
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Initializes the data value associated with a `DynValue` object from a sequence of name-`DynAny` pairs. Use this method instead of [set\\_members\(\)](#) if you want to avoid converting [DynAny](#) objects to `any` objects when your application needs to handle `DynValue` objects extensively.

### Parameters

`value`            A sequence of name-`DynAny` pairs representing member names and the values of the members as [DynAny](#) objects.

The current position is set to zero if the sequence passed in has a non-zero length. The current position is set to -1 if an empty sequence is passed in.

Members in the sequence must follow these rules:

- Members must be in the order in which they appear in the IDL specification of the structure.
- If member names are supplied in the sequence, they must either match the corresponding member name in the [TypeCode](#) of the `DynValue` or they must be empty strings.
- Members must be supplied in the same order as indicated by the [TypeCode](#) of the `DynValue`. The method does not reassign [DynAny](#) values based on member names.

### Exceptions

[InvalidValue](#)    The sequence has a number of elements that disagrees with the number of members as indicated by the [TypeCode](#) of the `DynValue`.

[TypeMismatch](#)    Raised if:

- One or more sequence elements have a type that is not equivalent to the [TypeCode](#) of the corresponding member.
- The member names do not match the corresponding member name in the [TypeCode](#) of the `DynValue`.

---

**See Also**

[DynamicAny::DynValue::get\\_members\\_as\\_dyn\\_any\(\)](#)

[DynamicAny::DynValue::set\\_members\(\)](#)

[DynamicAny::NameDynAnyPair](#)

# GSSUP Overview

The Generic Security Service username/password (GSSUP) IDL module defines the data types needed for the GSSUP mechanism. This reference page is an *extract* from the GSSUP module that includes only the data types needed for the `IT_CSI` module.

## GSSUP::GSSUPMechOID

```
const CSI::StringOID GSSUPMechOID = "oid:2.23.130.1.1.1";
```

The GSS Object Identifier allocated for the username/password mechanism, which is defined as follows:

```
{ iso-itu-t (2) international-organization (23) omg (130)
  security (1) authentication (1) gssup-mechanism (1) }
```

### See Also

```
IT_CSI::AuthenticationService::client_authentication_mech
IT_CSI::AuthenticationServicePolicy::client_authentication_mech
```

## GSSUP::ErrorCode

```
typedef unsigned long ErrorCode;
```

The error code type returned by GSSUP operations.

### See Also

```
IT_CSI::AuthenticateGSSUPCredentials::authenticate()
```

## GSSUP::GSS\_UP\_S\_G\_UNSPECIFIED

```
const ErrorCode GSS_UP_S_G_UNSPECIFIED = 1;
```

An error code indicating that the context validator has chosen not to reveal the GSSUP-specific cause of the failure.

### See Also

```
IT_CSI::AuthenticateGSSUPCredentials::authenticate()
```

---

## **GSSUP::GSS\_UP\_S\_G\_NOUSER**

```
const ErrorCode GSS_UP_S_G_NOUSER = 2;
```

An error code indicating that the user is unknown to the target.

### **See Also**

```
IT_CSI::AuthenticateGSSUPCredentials::authenticate()
```

## **GSSUP::GSS\_UP\_S\_G\_BAD\_PASSWORD**

```
const ErrorCode GSS_UP_S_G_BAD_PASSWORD = 3;
```

An error code indicating that the supplied password was incorrect.

### **See Also**

```
IT_CSI::AuthenticateGSSUPCredentials::authenticate()
```

## **GSSUP::GSS\_UP\_S\_G\_BAD\_TARGET**

```
const ErrorCode GSS_UP_S_G_BAD_TARGET = 4;
```

An error code indicating that the *target name*, by which is meant a security policy domain (CSlv2 authentication domain), does not match a security policy domain in the target.

### **See Also**

```
IT_CSI::AuthenticateGSSUPCredentials::authenticate()
```



# The IT\_Buffer Module

---

**In this chapter**

This chapter contains the following sections:

<a href="#">Module IT_Buffer</a>	<a href="#">page 772</a>
<a href="#">Interface IT_Buffer::Storage</a>	<a href="#">page 773</a>
<a href="#">Interface IT_Buffer::Segment</a>	<a href="#">page 775</a>
<a href="#">Interface IT_Buffer::Buffer</a>	<a href="#">page 777</a>
<a href="#">Interface IT_Buffer::BufferManager</a>	<a href="#">page 782</a>

---

# Module `IT_Buffer`

**Summary** A proprietary implementation of a segmented buffer, for use in ART-based applications.

**Description** ART Buffers are not expected to maintain storage in a contiguous region of memory. Instead Buffers are made up of Segments and, where appropriate, are optimized for bulk access to these Segments. Segments, in turn, each represent a subrange of the data contained in a Storage instance. Storage instances can be shared by multiple Buffer instances, allowing messages to be parsed without copying.

---

## `IT_Buffer::RawData`

**Summary** An IDL native type providing efficient access to a Buffer's data.

**Description** The `RawData` type provides access to a contiguous subset of the bytes contained in a `Buffer`. It is an IDL native type that maps to the language specific type that provides the most efficient access for marshaling and demarshaling individual primitives as well as for accessing bulk data.

**Java implementation** In Java, `RawData` maps to `byte[]` by default. If you are using JDK1.4 and you have enabled Orbix to use Java's new I/O (NIO), the `RawData` type maps to `java.nio.ByteBuffer` instead.

---

## `IT_Buffer::StorageSeq`

**Summary** A sequence of local `IT_Buffer::Storage` objects.

---

## Interface `IT_Buffer::Storage`

<b>Summary</b>	A contiguous region of bytes of which subranges can be contained in <code>Buffers</code> .
<b>Description</b>	<p>The ART core provides a heap-based <code>Storage</code> implementation. Plug-ins may provide special purpose <code>Storage</code> implementations—for example, referencing shared memory.</p> <p>Instances of <code>Storage</code> must be safe to access concurrently, because they might be contained as <code>SegmentS</code> in multiple <code>Buffers</code>.</p>
<b>Java implementation</b>	Because <code>Storage</code> instances potentially occupy significant amounts of scarce resources, Java garbage collection cannot be relied upon to reclaim instances that are no longer needed in a timely manner. Instead, <code>reference()</code> and <code>unreference()</code> operations are provided to explicitly maintain a count of how many <code>Buffer Segments</code> reference a <code>Storage</code> instance.

---

### `IT_Buffer::Storage::data`

<b>Summary</b>	Provides access to the bytes in the <code>Storage</code> object.
----------------	--

---

### `IT_Buffer::Storage::length`

<b>Summary</b>	The number of bytes in <code>IT_Buffer::Storage::data</code> .
----------------	--

---

### `IT_Buffer::Storage::another()`

<b>Summary</b>	Obtain another <code>Storage</code> instance of the same implementation type, and sharing any other relevant traits.
<b>Returns</b>	An otherwise unused <code>Storage</code> instance.
<b>Parameters</b>	<p><code>expiry</code></p> <p>Latest time at which to give up. The <code>Storage</code> implementation is free to impose a stricter expiry, for example for resource management when more one call to <code>another()</code> is in progress.</p>

**Exceptions**

CORBA: :TIMEOUT

Raised if an appropriate *Storage* instance cannot be obtained before expiry.

CORBA: :NO\_RESOURCES

Raised if the operation gives up before the specified expiry time.

---

**IT\_Buffer::Storage::reference()**

**Summary**

Increments the *Storage* instance's reference count.

**Java implementation**

This method should be called whenever you assign the *Storage* instance to a new Java reference.

---

**IT\_Buffer::Storage::unreference()**

**Summary**

Decrement the *Storage* instance's reference count.

**Java implementation**

This method should be called whenever a particular Java reference is finished using the *Storage* instance. When the reference count becomes zero, the *Storage* instance becomes available for re-use.

---

## Interface `IT_Buffer::Segment`

<b>Summary</b>	A contiguous subset of the data contained in a <code>Buffer</code> .
<b>Description</b>	A <code>Segment</code> represents a contiguous subset of the bytes contained in a <code>Buffer</code> . <code>Segments</code> are implemented by the ART core. <code>Segment</code> instances belong to a specific <code>Buffer</code> instance and are not reference counted in C++. <code>Segment</code> instances must be protected from concurrent access. The data attribute may expose bytes that belong to other <code>Segments</code> , which must not be examined or modified via this <code>Segment</code> .

---

### `IT_Buffer::Segment::data`

<b>Summary</b>	A reference to the block of raw memory where this segment is stored.
<b>Java implementation</b>	In Java, the native <code>RawData</code> type maps to <code>byte[]</code> by default. If you are using JDK1.4 and you have enabled Orbix to use Java's new I/O (NIO), the <code>RawData</code> type maps to <code>java.nio.ByteBuffer</code> instead.

---

### `IT_Buffer::Segment::offset`

<b>Summary</b>	The offset in <code>IT_Buffer::Segment::data</code> at which this <code>Segment</code> 's bytes begin.
<b>Description</b>	In other words, the first byte in this <code>Segment</code> is given by <code>Segment::data + Segment::offset</code> .

---

### `IT_Buffer::Segment::length`

<b>Summary</b>	The number of bytes in <code>IT_Buffer::Segment::data</code> that belong to this <code>Segment</code> .
<b>Description</b>	The value of <code>length</code> is always greater than zero. For example, the index after the last byte in the segment is given by <code>Segment::data + Segment::offset + Segment::length</code> .

## **IT\_Buffer::Segment::underlying\_storage**

### **Summary**

Returns the underlying storage as an `IT_Buffer::Storage` object.

---

## Interface `IT_Buffer::Buffer`

<b>Summary</b>	A randomly accessible linear finite sequence of bytes.
<b>Description</b>	A <code>Buffer</code> is made up of an ordered set of <code>Segments</code> , each providing access to a contiguous subrange of the <code>Buffer</code> 's data. <code>Buffers</code> are implemented by the ART core, and instances must be protected from concurrent access.
<b>Java implementation</b>	Because <code>Buffers</code> potentially occupy significant amounts of scarce resources, Java garbage collection cannot be relied upon to reclaim <code>Buffers</code> that are no longer needed in a timely manner. Instead, the <code>recycle()</code> operation must be called when a <code>Buffer</code> is no longer needed. Note that many operations to which <code>Buffers</code> are passed implicitly take responsibility for recycling those <code>Buffers</code> .

---

### `IT_Buffer::Buffer::length`

<b>Summary</b>	The number of bytes within the <code>Buffer</code> currently available for use.
----------------	---

---

### `IT_Buffer::Buffer::original_length`

<b>Summary</b>	The number of bytes originally allocated to the <code>Buffer</code> .
----------------	---

---

### `IT_Buffer::Buffer::storage_size`

<b>Summary</b>	The allocation unit size of the <code>Buffer</code> 's underlying <code>Storage</code> implementation.
----------------	--

---

### `IT_Buffer::Buffer::segment_count`

<b>Summary</b>	The number of segments currently available for use.
----------------	---

---

### `IT_Buffer::Buffer::rewind()`

<b>Summary</b>	Ensures that a subsequent call to <code>next_segment()</code> will return the first segment of the <code>Buffer</code> , or <code>NULL</code> if the length is zero.
----------------	--

---

## `IT_Buffer::Buffer::next_segment()`

<b>Summary</b>	Gets the next <code>Segment</code> of the <code>Buffer</code> .
<b>Description</b>	The first call to <code>next_segment()</code> after a <code>Buffer</code> has been allocated or <code>rewind()</code> has been called returns the first <code>Segment</code> of the <code>Buffer</code> . A subsequent call returns the <code>Segment</code> following the <code>Segment</code> that was previously returned.
<b>Returns</b>	The next segment, or <code>NULL</code> if the <code>Buffer</code> contains no additional segments.

---

## `IT_Buffer::Buffer::grow()`

<b>Summary</b>	Attempts to increase the length of the <code>Buffer</code> .
<b>Description</b>	On successful return, the <code>Buffer</code> 's length will have increased by at least <code>increment</code> bytes. It may be larger, if adding an integral number of <code>Storage</code> instances results in more than the requested number of bytes. If the most recent call to <code>next_segment()</code> had returned <code>NULL</code> , a call subsequent to a successful <code>grow()</code> by a non-zero increment will return the first newly added <code>Segment</code> .
<b>Parameters</b>	<code>increment</code> The minimum by which to increase the length. <code>expiry</code> Latest time at which to give up. The <code>Buffer</code> implementation is free to impose a stricter expiry time.
<b>Exceptions</b>	<code>CORBA::TIMEOUT</code> Raised if the <code>Buffer</code> cannot be grown to at least <code>new_length</code> bytes before expiry. <code>CORBA::NO_RESOURCES</code> Raised if the operation gives up before the specified expiry time.

---

## `IT_Buffer::Buffer::trim()`

<b>Summary</b>	Reduce the length, unreferencing any unneeded <code>Storage</code> instances.
<b>Description</b>	Trim always rewinds the <code>Buffer</code> .



<b>Parameters</b>	<p><i>from</i> The index of the first byte to be included in the trimmed <i>Buffer</i>.</p> <p><i>to</i> The index after the last byte to be included in the trimmed <i>Buffer</i>.</p>
<b>Exceptions</b>	<p>CORBA::BAD_PARAM Raised if an invalid subrange is specified.</p>

---

## IT\_Buffer::Buffer::eclipse()

<b>Summary</b>	Hides or exposes an initial subrange of the <i>Buffer</i> data.
<b>Description</b>	Nested eclipsing is allowed. The <i>Buffer</i> is always rewound by this operation.
<b>Parameters</b>	<p><i>delta</i> Specifies the offset from the current <i>Buffer</i> start index to hide (when positive) or expose (when negative)</p>
<b>Exceptions</b>	<p>CORBA::BAD_PARAM Raised if <i>delta</i> is outside the unecipsed buffer.</p>

---

## IT\_Buffer::Buffer::recycle()

<b>Summary</b>	Returns the <i>Buffer</i> to the <i>BufferManager</i> 's pool of unallocated <i>Buffer</i> s, unreferencing any <i>Storage</i> instances it contains.
<b>Exceptions</b>	<p>CORBA::BAD_INV_ORDER Raised if the buffer is already recycled.</p>

---

## IT\_Buffer::Buffer::prepend()

<b>Summary</b>	Concatenates another <i>Buffer</i> with this <i>Buffer</i> .
<b>Description</b>	The contents of the <i>head</i> is inserted prior to the current first byte of this <i>Buffer</i> . The <i>head</i> <i>Buffer</i> is implicitly recycled.
<b>Parameters</b>	<p><i>head</i> The other <i>Buffer</i>.</p>

---

## IT\_Buffer::Buffer::append()

<b>Summary</b>	Concatenates this <code>Buffer</code> with another <code>Buffer</code> .
<b>Description</b>	The contents of the <code>tail</code> is inserted after the current last byte of this <code>Buffer</code> . The tail <code>Buffer</code> is implicitly recycled. If the most recent call to <code>next_segment()</code> had returned <code>NULL</code> , a call subsequent to the <code>append()</code> of a non-empty buffer returns the first appended segment.
<b>Parameters</b>	<code>tail</code> The other <code>Buffer</code> .

---

## IT\_Buffer::Buffer::extract()

<b>Summary</b>	Extracts the specified range of bytes from this <code>Buffer</code> .
<b>Description</b>	The specified range of bytes are returned as a new <code>Buffer</code> . This <code>Buffer</code> is left containing the concatenation of the bytes before and after the specified range. Both this <code>Buffer</code> and the result are rewind.
<b>Returns</b>	A new <code>Buffer</code> containing the extracted bytes.
<b>Parameters</b>	<code>from</code> The index of the first byte to extract. <code>to</code> The index after the last byte to extract.
<b>Exceptions</b>	<code>CORBA::BAD_PARAM</code> Raised if an invalid subrange is specified.

---

## IT\_Buffer::Buffer::copy\_octets()

<b>Summary</b>	Copy a sub-range of the <code>Buffer</code> into an octet sequence.
<b>Parameters</b>	<code>buffer_offset</code> The offset into the <code>Buffer</code> to copy from. <code>dest</code> The destination octet sequence. The octets in the given sequence object can be modified, but the implementation should <i>not</i> return a different sequence.

`dest_offset`

The offset into the destination to copy into.

`length`

The number of bytes to copy.

## Exceptions

`CORBA::BAD_PARAM`

Raised if an invalid sub-range of the `Buffer` is specified.

---

## Interface IT\_Buffer::BufferManager

<b>Summary</b>	A per-ORB singleton object for managing Buffers.
<b>Description</b>	An instance of <code>BufferManager</code> is provided by the ART core, and is obtained by resolving the <code>IT_BufferManager</code> initial reference string.

---

### IT\_Buffer::BufferManager::get\_buffer()

<b>Summary</b>	Allocate a <code>Buffer</code> containing a single <code>Segment</code> that references the specified range of the specified <code>Storage</code> instance.
<b>Returns</b>	The newly allocated <code>Buffer</code> .
<b>Parameters</b>	<code>initial_segment_storage</code> The <code>Storage</code> object backing the initial segment. <code>initial_segment_offset</code> The offset in <code>initial_segment_storage</code> at which the initial segment begins. <code>initial_segment_length</code> The number of bytes in <code>initial_segment_storage</code> belonging to the initial segment.

---

### IT\_Buffer::BufferManager::get\_segmented\_buffer()

<b>Summary</b>	Allocates a <code>Buffer</code> containing a sequence of <code>Segments</code> , each backed by the corresponding member of the provided sequence of <code>Storages</code> , bounded by the relevant members of the offsets and lengths sequences.
<b>Description</b>	Typically used by a wrapping <code>Buffer</code> implementation.
<b>Returns</b>	The newly allocated <code>Buffer</code> .
<b>Parameters</b>	<code>storages</code> The sequence of <code>Storage</code> objects. <code>offsets</code> The sequence of offsets. <code>lengths</code> The sequence of lengths.

---

## IT\_Buffer::BufferManager::get\_heap\_buffer()

<b>Summary</b>	Allocate a <code>Buffer</code> containing the specified amount of heap-allocated Storage.
<b>Returns</b>	The newly allocated <code>Buffer</code> .
<b>Parameters</b>	<code>length</code> The number of bytes required; or zero, indicating a single <code>Segment</code> of the heap's preferred size.

---

## IT\_Buffer::BufferManager::get\_octets\_buffer()

<b>Summary</b>	Allocate a <code>Buffer</code> referencing an octet sequence's data.
<b>Returns</b>	The newly allocated <code>Buffer</code> .
<b>Parameters</b>	<code>octets</code> The octet sequence <code>offset</code> The offset into the octet sequence. <code>length</code> The number of octets to use.



# IT\_Certificate Overview

The `IT_Certificate` module provides data types and interfaces that are used to manage and describe X.509 certificates. The following interfaces are provided in this module:

- [Certificate](#)
- [X509Cert](#)
- [X509CertificateFactory](#)

## IT\_Certificate::CertError Exception

```
// IDL
exception CertError
{
    Error e;
};
```

A certificate-related error.

## IT\_Certificate::DERData Sequence

```
typedef sequence<octet> DERData;
```

Holds data in distinguished encoding rules (DER) format.

## IT\_Certificate::Error Structure

```
struct Error
{
    Error\_code err_code;
    string error_message;
};
```

Holds certificate-related error information.

---

## **IT\_Certificate::Error\_code Type**

```
typedef short Error_code;
```

Holds the certificate-related error codes.

### **Values**

This type can have one of the following integer constant values:

```
IT_TLS_FAILURE  
IT_TLS_UNSUPPORTED_FORMAT  
IT_TLS_BAD_CERTIFICATE_DATA  
IT_TLS_ERROR_READING_DATA
```

## **IT\_Certificate::X509CertChain Sequence**

```
typedef sequence<X509Cert> X509CertChain;
```

A list of `x509Cert` object references.

## **IT\_Certificate::X509CertList Sequence**

```
typedef sequence<X509Cert> X509CertList;
```

A list of `x509Cert` object references.



# IT\_Certificate::AVA Interface

## IDL

```
// IDL in module IT\_Certificate
interface AVA
{
    readonly attribute UShort set;
    readonly attribute ASN\_OID oid;

    // raises minor code IT_TLS_UNSUPPORTED_FORMAT
    Bytes convert(in Format f) raises(CertError);
};
```

Individual [AVA](#) objects represent an element of the distinguished name such as the common name field (CN) or organization unit (OU). You can retrieve a desired [AVA](#) object can using the [AVAList](#) class.

AVA objects can be converted to a number of different forms such as string format or DER format.

## AVA::convert()

```
// IDL
Bytes convert(in Format f) raises(CertError);
```

## Description

This operation returns the contents of the [AVA](#) object in the requested data format.

---

**Parameters**

This operation takes the following parameter

`f` The format of the required conversion. The following [Format](#) values are supported:

`IT_FMT_DER`. This format corresponds to the DER encoding of the AVA. This option is usually only used by applications that require special processing of the DER data.

`IT_FMT_STRING`. This format corresponds to a null-terminated sequence of characters containing the actual data of the AVA. The data is not modified in any way, and can include non-printable characters if present in the actual AVA data. This is a string for normal printable string fields.

`IT_FMT_HEX_STRING`. This format corresponds to a formatted hexadecimal dump of the DER data of the AVA.

**Exceptions**

[CertError](#) with An unknown format is specified.  
error code

`IT_TLS_UNSUPPO`  
`RTED_FORMAT`

**AVA::oid**

```
// IDL  
readonly attribute ASN\_OID oid;
```

**Description**

Return the ASN.1 OID tag for this AVA object, in the form of an `ASN_OID` structure.

**AVA::set**

```
// IDL  
readonly attribute UShort set;
```

---

**Description**

A number that identifies the set to which the AVA belongs. Because a set normally contains just a single AVA, the number returned by the `set` attribute is usually distinct for each AVA.

Theoretically, more than one AVA could belong to the same set, in which case two or more AVAs could share the same `set` number. In practice, this rarely ever happens.



# IT\_Certificate::AVAList Interface

## IDL

```
// IDL in module IT\_Certificate
interface AVAList
{
    typedef sequence<AVA> ListOfAVAs;
    readonly attribute ListOfAVAs ava_list;

    UShort get_num_avas();

    // Returns SUCCESSFUL or AVA_NOT_PRESENT
    IT\_Certificate::ReplyStatus
    get_ava_by_oid_tag(
        in OIDTag t,
        out AVA a
    ) raises(CertError);

    // Returns SUCCESSFUL or AVA_NOT_PRESENT
    IT\_Certificate::ReplyStatus
    get_ava_by_oid(
        in ASN\_OID seq,
        in UShort n,
        out AVA a
    ) raises(CertError);

    // raises minor code IT_TLS_UNSUPPORTED_FORMAT
    Bytes convert(
        in Format f
    ) raises(CertError);
};
```

## Description

An `AVAList` is an abstraction of a distinguished name from a certificate. An `AVAList` consists of a number of `AVA` objects.

Individual `AVA` objects represent an element of the distinguished name such as the common name field (CN) or organization unit (OU). You can retrieve a desired `AVA` object using the `AVAList`.

`AVA` objects can be converted to a number of different forms such as string format or DER format.

---

## AVAList::ava\_list

**IDL** readonly attribute ListOfAVAs ava\_list;

**Description** Returns the AVA list as a sequence of [AVA](#) object references.

## AVAList::convert()

**IDL** [Bytes](#) convert(in [Format](#) f) raises ([CertError](#));

**Description** This operation converts the AVAList to a specified format.

**Parameters** This operation takes the following parameter:

- f The format of the required conversion. The following `Format` values are supported:
- `IT_FMT_DER`. This format corresponds to the DER encoding of the AVA. This option is usually only used by applications that require special processing of the DER data.
  - `IT_FMT_STRING`. This format corresponds to a null-terminated sequence of characters containing the actual data of the AVA. The data is not modified in any way, and can include non-printable characters if present in the actual AVA data. This is a string for normal printable string fields.
  - `IT_FMT_HEX_STRING`. This format corresponds to a formatted hexadecimal dump of the DER data of the AVA.

## Exceptions

[CertError](#), An unknown format is specified.  
error code  
`IT_TLS_UNSUPPO`  
`RTED_FORMAT`

---

## AVAList::get\_ava\_by\_oid\_tag()

**IDL**

```
// Returns SUCCESSFUL or AVA_NOT_PRESENT
IT_Certificate::ReplyStatus
get_ava_by_oid_tag(
    in OIDTag t,
    out AVA a
) raises(CertError);
```

**Description** This operation retrieves an [AVA](#) object from an `AVAList` according to its OID tag.

### Parameters

t	An OID tag
a	The returned AVA object reference.

## AVAList::get\_ava\_by\_oid()

**IDL**

```
// Returns SUCCESSFUL or AVA_NOT_PRESENT
IT_Certificate::ReplyStatus
get_ava_by_oid(
    in ASN_OID seq,
    in UShort n,
    out AVA a
) raises(CertError);
```

**Description** This operation retrieves an [AVA](#) object from an `AVAList`, selected by the specified `ASN_OID` structure.

### Parameters

seq	An ASN OID.
n	
a	The returned AVA object reference.

## AVAList::get\_num\_avas()

**IDL**

```
UShort get_num_avas();
```

---

**Description** This operation retrieves the number of [AVA](#) objects in a `AVAList`.



# IT\_Certificate::Certificate Interface

**IDL**           // IDL in module [IT\\_Certificate](#)  
interface Certificate  
{  
    readonly attribute [DERData](#) encoded\_form;  
};

**Description**   This is the base interface for security certificate objects.

## **Certificate::encoded\_form**

**IDL**           readonly attribute [DERData](#) encoded\_form;

**Description**   This attribute returns the certificate data encoded in DER format.



# IT\_Certificate::Extension Interface

**IDL**

```
// IDL in module IT\_Certificate
interface Extension
{
    readonly attribute UShort critical;
    readonly attribute ASN\_OID oid;

    // raises minor code IT_TLS_UNSUPPORTED_FORMAT
    Bytes convert(in Format f) raises(CertError);
};
```

**Description** The `Extension` interface provides the developer with an interface to any X.509 version 3.0 extensions that an X.509 certificate can contain.

The `Extension` interface enables you to access the data for one particular extension. Using the `Extension::convert()` operations, the data can be converted into a number of representations.

## **Extension::convert()**

**IDL** [Bytes](#) convert(in [Format](#) f) raises([CertError](#));

**Description** This operation returns data that corresponds to the contents of the `Extension` object converted to the requested format. The data is converted to the requested format and returned as an array of bytes.

---

**Parameters**

This operation takes the following parameter:

- f The format of the required conversion. The following `Format` values are supported:

`IT_FMT_DER`. This format corresponds to the DER encoding of the extension. This option is usually only used by applications that require special processing of the DER data.

`IT_FMT_STRING`. This format corresponds to a null terminated sequence of characters containing the actual data contained in the extension. This data has not been modified in any way, and may include non printable characters if present in the actual extension data. This is a regular 'C' string for printable string fields.

`IT_FMT_HEX_STRING`. This format contains a formatted hexadecimal dump of the DER data of the extension.

### **Extension::critical**

**IDL**

readonly attribute [UShort](#) critical;

**Description**

This attribute returns a non-zero value if the extension is critical; zero if the extension is not critical. A critical extension is an extension that should not be ignored by the authentication code.

### **Extension::oid**

**IDL**

readonly attribute [ASN\\_OID](#) oid;

**Description**

This attribute returns the ASN.1 OID for the extension. Extensions are identified by an ASN.1 OID, just like regular AVAs.

# IT\_Certificate::ExtensionList Interface

## IDL

```
// IDL in module IT\_Certificate
interface ExtensionList
{
    typedef sequence<Extension> ListOfExtensions;
    readonly attribute ListOfExtensions    ext_list;

    UShort get_num_extensions();

    // Returns SUCCESSFUL or EXTENSION_NOT_PRESENT
    IT\_Certificate::ReplyStatus
    get_extension_by_oid_tag(
        in OIDTag t,
        out Extension e
    ) raises(CertError);

    // Returns SUCCESSFUL or EXTENSION_NOT_PRESENT
    IT\_Certificate::ReplyStatus
    get_extension_by_oid(
        in ASN\_OID seq,
        in UShort n,
        out Extension e
    ) raises(CertError);

    // raises minor code IT_TLS_UNSUPPORTED_FORMAT
    Bytes convert(in Format f) raises(CertError);
};
```

## Description

The [Extension](#) and `ExtensionList` interfaces provide you with access to any X.509 version three extensions.

The `Extension` interface provides an interface to accessing the data for one particular extension.

## ExtensionList::convert()

## IDL

```
Bytes convert(in Format f) raises(CertError);
```

---

**Description** `convert()` returns data in the requested format corresponding to the contents of the `ExtensionList` object. The operation returns this data as an array of bytes, or NULL if the the required conversion is not supported.

---

**Note:** Generally `convert()` is called on the individual extensions. This operation is not commonly used.

---

**Parameters** This operation takes the following parameter:

`f` The format of the required conversion. The following `Format` value is supported:

`IT_FMT_DER`. This format corresponds to the DER encoding of the AVA. This option is usually only used by applications that require special processing of the DER data.

`IT_FMT_STRING`. This format corresponds to a null-terminated sequence of characters containing the actual data of the AVA. The data is not modified in any way, and can include non-printable characters if present in the actual AVA data. This is a string for normal printable string fields.

`IT_FMT_HEX_STRING`. This format corresponds to a formatted hexadecimal dump of the DER data of the AVA.

## Exceptions

[CertError](#), An unknown format is specified.  
error code  
`IT_TLS_UNSUPPO`  
`RTED_FORMAT`

## `ExtensionList::ext_list`

**IDL** `readonly attribute ListOfExtensions ext_list;`

**Description** This attribute returns the complete list of extensions as a sequence of [Extension](#) objects.

---

## ExtensionList::get\_extension\_by\_oid()

### IDL

```
IT\_Certificate::ReplyStatus  
get_extension_by_oid(  
    in ASN\_OID seq,  
    in UShort n,  
    out Extension e  
) raises(CertError);
```

### Description

Obtains the [Extension](#) element of the `ExtensionList` that has the requested object identifier, `seq`.

If the extension is found, a `SUCCESSFUL` reply status is returned; otherwise an `EXTENSION_NOT_PRESENT` reply status is returned.

### Parameters

This operation takes the following parameters

`seq`        An array of integers representing the ASN.1 object identifier.  
`n`            The number of elements in the array.  
`e`            The returned `Extension` object.

## ExtensionList::get\_extension\_by\_oid\_tag()

### IDL

```
IT\_Certificate::ReplyStatus  
get_extension_by_oid_tag(  
    in OIDTag t,  
    out Extension e  
) raises(CertError);
```

### Description

Obtains the [Extension](#) element of the `ExtensionList` that corresponds to the supplied [OIDTag](#) value, `t`.

If the extension is found, a `SUCCESSFUL` reply status is returned; otherwise an `EXTENSION_NOT_PRESENT` reply status is returned.

### Parameters

`t`            The `OIDTag` variable that identifies the extension to retrieve.  
`e`            The returned `Extension` object.

---

## ExtensionList::get\_num\_extensions();

**IDL** [UShort](#) get\_num\_extensions();

**Description** This operation returns the number of extensions in the list.



# IT\_Certificate::X509Cert Interface

**IDL**

```
// IDL in module IT\_Certificate
interface X509Cert : IT\_Certificate::Certificate
{
    // The only X509Cert definition supported in
    // Java is the inherited
    // IT_Certificate::Certificate::encoded_form attribute.
};
```

**Description** The `IT_Certificate::X509Cert` is the type of object returned by certain operations and attributes in the `IT_TLS_API` module. The only operation or attribute supported by the `X509Cert` interface is the `encoded_form` attribute, which is inherited from `IT_Certificate::Certificate`.

To access the information in a Java X.509 certificate, it is recommended that you perform the following steps:

1. Extract the DER data from the certificate using the `IT_Certificate::Certificate::encoded_form` attribute.
2. Pass the DER data to the `com.ionacorba.tls.cert.CertHelper.bytearray_to_cert()` method to obtain a `java.security.cert.Certificate` object.
3. Use the `java.security.cert` package to examine the certificate.



# IT\_Certificate::X509CertificateFactory Interface

**IDL**

```
// IDL in module IT\_Certificate
interface X509CertificateFactory
{
    // Following function creates x509Cert from DER data.
    // where DERData is a sequence of octets
    //
    // raises minor code IT_TLS_BAD_CERTIFICATE_DATA
    //
    X509Cert
    create_x509_certificate_from_der(
        in DERData der
    ) raises(CertError);

    //
    // Read CertList from a file.
    // raises minor code IT_TLS_BAD_CERTIFICATE_DATA.
    // raises minor code IT_TLS_ERROR_READING_DATA.
    //
    X509CertList
    load_x509_cert_list(
        in string location
    ) raises(CertError);
};
```

## Description

This interface is a factory that generates X.509 certificates of [IT\\_Certificate::X509Cert](#) type.

This interface contains one operation, `create_x509_cert()`, that generates an X.509 certificate on receiving data in the form of DER.

## X509CertificateFactory::create\_x509\_certificate\_from\_der()

**IDL** [X509Cert](#)

---

```
create_x509_certificate_from_der(  
    in DERData der  
    ) raises(CertError);
```

**Description** Generates an X.509 certificate based on a parameter supplied in DER format, `der`.

**Parameters** This operation takes the following parameter:

`der`            The certificate data in DER format (of `DERData` type).

**Exceptions**

[CertError](#),     The `der` parameter is inconsistent or incorrectly formatted  
error code  
`IT_TLS_BAD_CER  
TIFICATE_DATA`

## **X509CertificateFactory::load\_x509\_cert\_list()**

**IDL**

```
X509CertList  
load_x509_cert_list(in string location) raises(CertError);
```

**Description** Generates a list of X.509 certificates based on data read from the file specified by `location`. The file must contain a chain of certificates in PEM format.

**Parameters** This operation takes the following parameter:

`location`        The absolute path name of the file containing the PEM certificate chain.

---

## Exceptions

[CertError](#), Orbix cannot read the specified certificate file  
error code  
IT\_TLS\_ERRO  
R\_READING\_D  
ATA

[CertError](#), The content of the certificate file is inconsistent or incorrectly  
error code formatted.  
IT\_TLS\_BAD\_  
CERTIFICATE  
\_DATA



# IT\_Config Overview

Every ORB is associated with a configuration domain that provides it with configuration information. The configuration mechanism enables Orbix to get its configuration information from virtually any source including files or configuration repositories. The `IT_Config` module contains the API to both get configuration settings and receive notifications when a particular configuration value changes. The module contains the following interfaces:

- [Configuration](#)
- [Listener](#)

The `IT_Config` module does not give you a mechanism for changing configurations. Administrators typically setup and manage a configuration domain using various tools described in the *Application Server Platform Administrator's Guide*.

A single *configuration domain* can hold configuration information for multiple ORBs – each ORB uses its ORB name as a “key” to locate its particular configuration within the domain. Often, an administrator will want to use a default configuration domain for a group of applications, overriding only certain configuration variables for individual applications or ORBs. This might be useful within a hierarchical organization, or where different development groups or applications need slightly different configurations.

A configuration domain can be organized into a hierarchy of nested *configuration scopes* to enable a high degree of flexibility. Each scope within a domain must be uniquely named relative to its containing scope. Scope names consist of any combinations of alphanumeric characters and underscores. Scopes are usually identified by their fully qualified name, which contains the scope name and the names of all parent scopes, separated by a dot (.).

Within each configuration scope, variables are organized into configuration contexts. A *configuration context* is simply a collection of related configuration variables. A context may also contain sub-contexts. You can consider the configuration scope as the root context. Contained in the root context are a number of sub-contexts. For example, there is a plug-ins context and an initial-references context. The initial-references context contains a list of

---

initial-references for the services available to the system. The plug-ins context contains a sub-context for each plug-in, in which it holds its configuration information. This context will have the same name as the plug-in, and will hold information such as the name of the plug-in library and any dependencies the plug-in has, as well as other plug-in-specific settings.

You as a programmer need not worry about this configuration hierarchy set up by your administrator. You simply request configuration values via the `Configuration` interface. See the *Application Server Platform Administrator's Guide* for more on configuration.

## IT\_Config::ConfigList Sequence

```
// IDL
typedef sequence<string> ConfigList;
```

A list of configuration settings as strings.

**Enhancement** This is an Orbix enhancement.

**See Also** [IT\\_Config::Configuration::get\\_list\(\)](#)  
[IT\\_Config::Listener::list\\_changed\(\)](#)

## IT\_Config::ListenerTargetRange Enumeration

```
// IDL
enum ListenerTargetRange {
    OBJECT_ONLY,
    ONELEVEL,
    SUBTREE
};
```

A target scope refers to the extent of a configuration hierarchy that a [Listener](#) object monitors.

`OBJECT_SCOPE` The [Listener](#) is only interested in changes to the specific target variable. For example, a [Listener](#) with a target variable of `initial_references:Naming:reference` and a target scope of `OBJECT_SCOPE` is informed if that variable changes.



---

ONELEVEL_SCOPE	The <a href="#">Listener</a> is interested in changes to variables contained in the target, a configuration context, but not the target itself. For example, if the target is <code>plugins:iiop</code> , the <a href="#">Listener</a> is informed of any changes to variable in the <code>plugins:iiop</code> configuration context.
SUBTREE_SCOPE	The <a href="#">Listener</a> is interested in changes to the target and any variables or namespaces in the subtree of the target. For example, if the target is <code>initial_references</code> , the <a href="#">Listener</a> is informed of any changes to anything under the <code>initial_references</code> namespace, including the namespace itself.

**Enhancement** This is an Orbix enhancement.

**See Also** [IT\\_Config::Configuration::add\\_listener\(\)](#)



# IT\_Config::Configuration Interface

This interface provides access to configuration information. You get a reference to a `Configuration` implementation by calling `ORB::resolve_initial_references()` with the string argument `IT_Configuration`.

In a configuration domain, the ORB name acts as the configuration scope in which to start looking for configuration information. The ORB supplies this information when querying the configuration system for a configuration variable. If the variable cannot be found within that scope or the scope does not exist, the system recursively searches the containing scope. For example, if an ORB with an ORB name of `IONA.ProdDev.TestSuite.TestMgr` requests a variable, the system will first look in the `IONA.ProdDev.TestSuite.TestMgr` scope, then `IONA.ProdDev.TestSuite`, and so on, until it finally looks in the root scope. This allows administrators to place default configuration information at the highest level scope, then override this information in descendant scopes to produce a specific, tailored configuration.

Although there are specific operations such as `get_boolean()` and `get_double()` to retrieve certain types of configuration information, the `Configuration` interface is not strictly typed. This means that when a certain type of variable is requested, an effort is made to convert the retrieved value to the requested type. For example, if you call `get_long()`, and the domain has a string such as "1234", an attempt is made to convert the string to a long. In this case, it can successfully return 1234 as a long. If, however, the value for the requested variable were words such as "A String Value", then it cannot be converted to a long and a `TypeMismatch` exception is thrown.

```
// IDL in module IT_Config
interface Configuration {

    exception TypeMismatch {};

    boolean get\_string(
        in string name,
        out string value
    ) raises (TypeMismatch);
```

```
boolean get\_list(
    in string      name,
    out ConfigList value
) raises (TypeMismatch);

boolean get\_boolean(
    in string name,
    out boolean value
) raises (TypeMismatch);

boolean get\_long(
    in string name,
    out long value
) raises (TypeMismatch);

boolean get\_double(
    in string name,
    out double value
) raises (TypeMismatch);

void register\_listener(
    in string target,
    in ListenerTargetRange target_scope,
    in Listener l
);

void remove\_listener(
    in Listener l
);

// INTERNAL USE ONLY
//
void shutdown();
};
```

### **Configuration::register\_listener()**

```
// IDL
void register_listener(
    in string target,
```

---

```
        in ListenerTargetRange target_scope,  
        in Listener l  
    );
```

Adds a [Listener](#) object so your application can be notified of certain configuration changes.

### Parameters

`target`            The target configuration value for the [Listener](#).  
`target_scope`    The scope parameter determines the extent of change that the [Listener](#) is told about.  
`l`                 The [Listener](#) object.

Not all types of configuration domains support change notification.

**Enhancement** This is an Orbix enhancement.

**See Also**        [IT\\_Config::ListenerTargetRange](#)  
                 [IT\\_Config::Configuration::remove\\_listener\(\)](#)

### **Configuration::get\_boolean()**

```
// IDL  
boolean get_boolean(  
    in string name,  
    out boolean value  
) raises (TypeMismatch);
```

Returns true if the boolean value is successfully retrieved and false if the variable could not be found.

### Parameters

`name`             Name of the variable to retrieve.  
`value`            The value of the variable returned.

**Enhancement** This is an Orbix enhancement.

### Exceptions

[TypeMismatch](#)    The variable exists but is of the wrong type for this operation.

### Configuration::get\_double()

```
// IDL
boolean get_double(
    in string name,
    out double value
) raises (TypeMismatch);
```

Returns true if the double value is successfully retrieved and false if the variable could not be found.

#### Parameters

name	Name of the variable to retrieve.
value	The value of the variable returned.

**Enhancement** This is an Orbix enhancement.

#### Exceptions

[TypeMismatch](#) The variable exists but is of the wrong type for this operation.

### Configuration::get\_list()

```
// IDL
boolean get_list(
    in string name,
    out ConfigList value
) raises (TypeMismatch);
```

Returns true if the list of configuration settings is successfully retrieved and false if the list could not be found.

The semantics of this operation changed with the release of Orbix 6.2.

For the Java implementation of `get_list()`, [Table 14](#) shows the list of strings returned in the `value` parameter for a variety of different Orbix configuration settings.

**Table 14:** *Java Specifying Value Lists in Orbix Configuration*

	Orbix 6.1		Orbix 6.2	
	Length	Value list	Length	Values list
<code>list1 = "";</code>	0		0	
<code>list2 = [];</code>	0		0	
<code>list3 = [""]; </code>	0		1	" "
<code>list4 = ["", ""]; </code>	0		2	"", "
<code>list5 = ["a"]; </code>	1	"a"	1	"a"
<code>list6 = ["a", "b"]; </code>	2	"a", "b"	2	"a", "b"
<code>list7 = ["a", ""]; </code>	1	"a"	2	"a", "

Configuration lists can also be specified as command-line arguments. In Java, the syntax is a comma-separated list all contained within double quotes. The list of strings returned in the `value` parameter are as shown in [Table 15](#).

**Table 15:** *Java Specifying Value Lists in a Command-Line Parameter*

	Orbix 6.1		Orbix 6.2	
	Length	Value list	Length	Values list
<code>-ORblist1 ""</code>	0		0	
<code>-ORblist2 ", "</code>	1	", "	2	"", "
<code>-ORblist3 "a"</code>	1	"a"	1	"a"
<code>-ORblist4 "a,b"</code>	2	"a", "b"	2	"a", "b"
<code>-ORblist5 "a, "</code>	1	"a"	2	"a", "

### Parameters

name	Name of the configuration list to retrieve.
value	The values returned.

**Enhancement** This is an Orbix enhancement.

### Exceptions

[TypeMismatch](#) The variable exists but is of the wrong type for this operation.

### Configuration::get\_long()

```
// IDL
boolean get_long(
    in string name,
    out long value
) raises (TypeMismatch);
```

Returns true if the long value is successfully retrieved and false if the variable could not be found.

### Parameters

name	Name of the variable to retrieve.
value	The value of the variable returned.

**Enhancement** This is an Orbix enhancement.

### Exceptions

[TypeMismatch](#) The variable exists but is of the wrong type for this operation.

### Configuration::get\_string()

```
// IDL
boolean get_string(
    in string name,
    out string value
) raises (TypeMismatch);
```



---

Returns true if the string value is successfully retrieved and false if the variable could not be found.

For Orbix 6.2, the behavior of `get_string()` operation, when applied to a variable stored as a list in a configuration file, has changed in the Java ORB so that it conforms with the behavior it has always had in the C++ ORB, which is to throw an `IT_Config::Configuration::TypeMismatch` exception.

**Table 16:**

	Orbix 6.1		Orbix 6.2	
	C++	Java	C++	Java
<code>list5 = ["a"];</code>				
<code>get_string("list5")</code>	TypeMismatch	"a"	TypeMismatch	TypeMismatch

#### Parameters

`name` Name of the variable to retrieve.  
`value` The value of the variable returned.

**Enhancement** This is an Orbix enhancement.

#### Exceptions

[TypeMismatch](#) The variable exists but is of the wrong type for this operation.

### Configuration::remove\_listener()

```
// IDL
void remove_listener(
    in Listener l
);
```

Removes a [Listener](#) object.

**Enhancement** This is an Orbix enhancement.

**See Also** [IT\\_Config::Configuration::add\\_listener\(\)](#)

### **Configuration::shutdown()**

```
// IDL
void shutdown();
```

---

**Note:** For internal use only

---

### **Configuration::TypeMismatch Exception**

```
// IDL
exception TypeMismatch {};
```

The type of the configuration variable named in the operation does not match the type required for the operation.

**Enhancement** This is an Orbix enhancement.

# IT\_Config::Listener Interface

You can add a `Listener` object to your application that will be notified of configuration changes that occur. Use [add\\_listener\(\)](#) and [remove\\_listener\(\)](#) of the [Configuration](#) interface to manage a `Listener` object.

```
// IDL in module IT_Config
interface VariableListener : Listener {
    void variable_added(
        in string name
    );

    void variable_removed(
        in string name
    );

    void string\_changed(
        in string name,
        in string new_value,
        in string old_value
    );

    void list\_changed(
        in string name,
        in ConfigList new_value,
        in ConfigList old_value
    );

    void boolean\_changed(
        in string name,
        in boolean new_value,
        in boolean old_value
    );

    void long\_changed(
        in string name,
        in long new_value,
        in long old_value
    );
};
```

```
    );  
  
    void double\_changed(  
        in string name,  
        in double new_value,  
        in double old_value  
    );  
};
```

### **Listener::variable\_added()**

```
void variable_added(  
    in string name;  
)
```

The application is notified in a variable is added to the configuration.

#### **Parameters**

name                    The name of the variable added.

**Enhancement**    This is an Orbix enhancement.

### **Listener::variable\_removed()**

```
void variable_removed(  
    in string name;  
)
```

The application is notified in a variable is removed from the configuration.

#### **Parameters**

name                    The name of the variable removed.

**Enhancement**    This is an Orbix enhancement.

### **Listener::boolean\_changed()**

```
// IDL  
void boolean_changed(  

```

---

```
        in string name,  
        in boolean new_value,  
        in boolean old_value  
    );
```

The application is notified if the boolean value changes.

### Parameters

name	The name of the variable.
new_value	The value of the variable after the change occurred. If a variable is deleted this value will be NULL.
old_value	The previous value of the variable before the change occurred. If a variable is added this value will be NULL.

**Enhancement** This is an Orbix enhancement.

### Listener::double\_changed()

```
// IDL  
void double_changed(  
    in string name,  
    in double new_value,  
    in double old_value  
);
```

The application is notified if the double value changes.

### Parameters

name	The name of the variable.
new_value	The value of the variable after the change occurred. If a variable is deleted this value will be NULL.
old_value	The previous value of the variable before the change occurred. If a variable is added this value will be NULL.

**Enhancement** This is an Orbix enhancement.

### Listener::list\_changed()

```
// IDL
void list_changed(
    in string    name,
    in ConfigList new_value,
    in ConfigList old_value
);
```

The application is notified if the configuration list changes.

#### Parameters

name	The name of the variable.
new_value	The value of the variable after the change occurred. If a variable is deleted this value will be NULL.
old_value	The previous value of the variable before the change occurred. If a variable is added this value will be NULL.

**Enhancement** This is an Orbix enhancement.

### Listener::long\_changed()

```
// IDL
void long_changed(
    in string name,
    in long   new_value,
    in long   old_value
);
```

The application is notified if the long value changes.

#### Parameters

name	The name of the variable.
new_value	The value of the variable after the change occurred. If a variable is deleted this value will be NULL.
old_value	The previous value of the variable before the change occurred. If a variable is added this value will be NULL.

**Enhancement** This is an Orbix enhancement.

---

## Listener::string\_changed()

```
// IDL
void string_changed(
    in string name,
    in string new_value,
    in string old_value
);
```

The application is notified if the string value changes.

### Parameters

name	The name of the variable.
new_value	The value of the variable after the change occurred. If a variable is deleted this value will be NULL.
old_value	The previous value of the variable before the change occurred. If a variable is added this value will be NULL.

**Enhancement** This is an Orbix enhancement.





# IT\_CORBA Overview

This module contains Orbix enhancements to the [CORBA](#) module. The key additional feature is the policy `WellKnownAddressingPolicy`. The classes include:

- [WellKnownAddressingPolicy](#)

The IDL code is as follows:

## IT\_CORBA::WELL\_KNOWN\_ADDRESSING\_POLICY\_ID Constant

```
// IDL in module IT_CORBA
const CORBA::PolicyType WELL_KNOWN_ADDRESSING_POLICY_ID =
0x49545F00 + 2;
```

Defines a policy ID for well-known addressing.

**Enhancement** This is an Orbix enhancement to CORBA.

**See Also** [CORBA::PolicyType](#)



# IT\_CORBA::RefCountedLocalObject Class

`RefCountedLocalObject` is an implementation of a CORBA local object that automatically handles reference counting in a thread safe manner.

**See Also** [IT\\_CORBA::RefCountedLocalObjectNC](#)

## **RefCountedLocalObject::\_add\_ref()**

Increments the reference count.

**Enhancement** This is an Orbix enhancement to CORBA.

## **RefCountedLocalObject::\_destroy\_this()**

Destroys the local object.

**Enhancement** This is an Orbix enhancement to CORBA.

## **RefCountedLocalObject::RefCountedLocalObject() Constructor**

The constructor.

**Enhancement** This is an Orbix enhancement to CORBA.

## **RefCountedLocalObject::\_remove\_ref()**

Decrements the reference count.

**Enhancement** This is an Orbix enhancement to CORBA.



# IT\_CORBA:: RefCountedLocalObjectNC Class

RefCountedLocalObjectNC is an implementation of a CORBA local object that automatically handles reference counting but not in a thread-safe manner as the [RefCountedLocalObject](#) class does. A RefCountedLocalObjectNC object does not protect its reference count with a mutex, making it suitable for lightweight objects such as [CORBA::Request](#).

**See Also** [IT\\_CORBA::RefCountedLocalObject](#)

## RefCountedLocalObjectNC::\_add\_ref()

Increments the reference count.

**Enhancement** This is an Orbix enhancement to CORBA.

## RefCountedLocalObjectNC::\_destroy\_this()

Destroys the local object.

**Enhancement** This is an Orbix enhancement to CORBA.

## RefCountedLocalObjectNC::RefCountedLocalObjectNC() Constructor

The constructor.

**Enhancement** This is an Orbix enhancement to CORBA.

---

## **RefCountedLocalObjectNC::\_remove\_ref()**

Decrements the reference count.

**Enhancement** This is an Orbix enhancement to CORBA.

# IT\_CORBA:: WellKnownAddressingPolicy Class

This is an interface for a local policy object derived from [CORBA::Policy](#). You create instances of `WellKnownAddressingPolicy` by calling [CORBA::ORB::create\\_policy\(\)](#).

## `WellKnownAddressingPolicy::config_scope()`

```
// Java  
public java.lang.String config_scope()
```

Returns the configuration scope.

**Enhancement** This is an Orbix enhancement to CORBA.





# The IT\_CORBASEC Module

**In this chapter**

---

This chapter contains the following sections:

<a href="#">Module IT_CORBASEC</a>	<a href="#">page 836</a>
<a href="#">Interface IT_CORBASEC::ExtendedReceivedCredentials</a>	<a href="#">page 839</a>

---

# Module IT\_CORBASEC

**Summary** A module that gives you read/write access to extended received credentials.

**Description** In particular, the `IT_CORBASEC::ExtendedReceivedCredentials` interface gives you access to the received SSO tokens.

---

## IT\_CORBASEC::EXT\_ATTR\_ERR\_ATTR\_NOT\_PRESENT

**Summary** Raised by `get_extended_attribute()`, if the requested attribute is not present.

**Description** If this exception is raised, it implies that the requested attribute is neither present in the incoming request's service contexts nor has the requested attribute been set by a call to `IT_CORBASEC::ExtendedReceivedCredentials::set_extended_attribute()`.

---

## IT\_CORBASEC::EXT\_ATTR\_ERR\_FAILURE\_PROCESSING\_ATTR

**Summary** Not used.

---

## IT\_CORBASEC::EXT\_ATTR\_ERR\_READ\_ONLY\_ATTRIBUTE

**Summary** Raised by `set_extended_attribute()`, if the requested attribute is intended to be read-only.

**Description** Specifically, this error is raised if you attempt to set the `IT_CORBASEC::EXT_ATTR_CURRENT_SSO_TOKEN` attribute directly.

---

## IT\_CORBASEC::ExtendedAttributeError

**Summary** Exception raised by operations from the `IT_CORBASEC::ExtendedReceivedCredentials` interface.

**See also** `IT_CORBASEC::EXT_ATTR_ERR_ATTR_NOT_PRESENT`  
`IT_CORBASEC::EXT_ATTR_ERR_READ_ONLY_ATTRIBUTE`

---

## IT\_CORBASEC::SSOTokenString

<b>Summary</b>	Type of an SSO token.
<b>Description</b>	<p>An <code>SSOTokenString</code> can be extracted from the <code>any</code> returned from a call to <code>IT_CORBASEC::ExtendedReceivedCredentials::get_extended_attribute( )</code>, if the requested attribute is an SSO token.</p> <p>An <code>SSOTokenString</code> can be inserted into an <code>any</code> and passed in a call to <code>IT_CORBASEC::ExtendedReceivedCredentials::set_extended_attribute( )</code> to set an SSO token attribute.</p>
<b>See also</b>	<code>IT_CORBASEC::EXT_ATTR_CURRENT_SSO_TOKEN</code> <code>IT_CORBASEC::EXT_ATTR_DELEGATED_SSO_TOKEN</code> <code>IT_CORBASEC::EXT_ATTR_PEER_SSO_TOKEN</code>

---

## IT\_CORBASEC::EXT\_ATTR\_CURRENT\_SSO\_TOKEN

<b>Summary</b>	The attribute type for the current SSO token, which can be either a delegated token or a peer token.
<b>Description</b>	<p>The current SSO token is the token that would be used when making access control decisions for the incoming invocation. The value returned for the current SSO token can be one of the following (in order of priority):</p> <ul style="list-style-type: none"><li>• Delegated SSO token, if it is present, otherwise</li><li>• Peer SSO token, if it is present, otherwise</li><li>• No value.</li></ul>
<b>See also</b>	<code>IT_CORBASEC::EXT_ATTR_DELEGATED_SSO_TOKEN</code> <code>IT_CORBASEC::EXT_ATTR_PEER_SSO_TOKEN</code>

---

## IT\_CORBASEC::EXT\_ATTR\_DELEGATED\_SSO\_TOKEN

<b>Summary</b>	The attribute type for a delegated SSO token.
<b>Description</b>	In a multi-tier system (consisting of at least three tiers), a <i>delegated SSO token</i> represents a credential that originated at least two steps back in the invocation chain.

Currently, the only security mechanism in Orbix that supports delegation is CSI Identity Assertion.

The delegated token originates from a previous application in the invocation chain and is always copied into the effective credentials for the current execution context. Hence, in a multi-tiered system, the delegated SSO token received from the preceding application would automatically be used as the delegated credentials for the next invocation in the chain.

---

## IT\_CORBASEC::EXT\_ATTR\_PEER\_SSO\_TOKEN

### Summary

The attribute type for a peer SSO token.

### Description

A *peer SSO token* represents a credential that originates from the preceding application in the invocation chain and is received through the CSI authentication over transport mechanism.

A peer SSO token is available from an incoming request message on the server side, if the following conditions hold:

- Server is configured to use CSI authentication over transport.
- Client is configured to use CSI authentication over transport.
- Client is configured to use *either* username/password-based SSO *or* X.509 certificate-based SSO.

If there are no delegated credentials in the received credentials, the peer SSO token is used as the delegated credential in the current execution context. Hence, in the absence of received delegated credentials, the peer SSO token received from the preceding application is used as the delegated credentials for the next invocation in the chain.

---

## Interface `IT_CORBASEC::ExtendedReceivedCredentials`

<b>Summary</b>	An IONA specific interface that allows access to additional IONA specific logical attributes of a received credentials object.
<b>Description</b>	<p>An instance of a received credentials object is obtained by narrowing the received credentials object obtained from security current.</p> <p>The attribute IDs passed as arguments to the <code>get_extended_attribute()</code> and <code>set_extended_attribute()</code> operations are assigned by IONA. The range below 10000 is reserved for IONA use. These numbers are unique across all security mechanisms.</p>
<b>See also</b>	<a href="#">SecurityLevel2::Current</a> <a href="#">SecurityLevel2::ReceivedCredentials</a>

---

### `IT_CORBASEC::ExtendedReceivedCredentials::get_extended_attribute()`

<b>Summary</b>	Returns the value of a received credentials' extended attribute.
<b>Description</b>	<p>There are two possible origins of an extended attribute:</p> <ul style="list-style-type: none"><li>• From parsing a service context in the incoming request message.</li><li>• From a previous call to <code>set_extended_attribute()</code>, which set the attribute value on the received credentials object.</li></ul>
<b>Returns</b>	The value of an extended attribute contained in an <code>any</code> .
<b>Parameters</b>	<p><code>req_attribute</code></p> <p>An integer attribute ID, which identifies a particular extended attribute.</p>
<b>Exceptions</b>	<p><code>ExtendedAttributeError</code></p> <p>Raised with an <code>error_reason</code> of <code>EXT_ATTR_ERR_ATTR_NOT_PRESENT</code> if the requested attribute is not set.</p>

---

### `IT_CORBASEC::ExtendedReceivedCredentials::set_extended_attribute()`

<b>Summary</b>	Sets the value of a received credentials' extended attribute.
----------------	---

**Description**

The main purpose of setting an extended attribute is to influence subsequent remote CORBA invocations within the current execution context. The received credentials can affect subsequent invocations, because Orbix takes received credentials into account when creating the effective credentials for a new invocation.

For example, if a delegated SSO token attribute is set in the received credentials, it would automatically be copied into the effective credentials for a new invocation (by the GSP plug-in).

**Parameters**

`attribute_type`

An integer attribute ID, which identifies a particular extended attribute.

`any_val`

The value of an extended attribute contained in an `any`.

**Exceptions**

`ExtendedAttributeError`

Raised with an `error_reason` of `EXT_ATTR_ERR_READ_ONLY_ATTRIBUTE` if the requested attribute is not intended to be settable.

# IT\_CosTransactions Module

The `IT_CosTransactions` module contains Orbix 2000 enhancements to the standard OTS [CosTransactions](#) module. The `IT_CosTransactions` module includes additional values for the [OTSPolicyValue](#) data type and proprietary extensions to the standard [CosTransactions::Current](#) class.

## Additional OTSPolicyValues

```
const OTSPolicyValue AUTOMATIC = 4;  
const OTSPolicyValue SERVER_SIDE = 5;
```

These additional `OTSPolicyValue`s indicate the following:

- |                          |  |
|--------------------------|--|
| <code>AUTOMATIC</code>   | The target object depends on the presence of a transaction. If there is no current transaction, a transaction is created for the duration of the invocation. |
| <code>SERVER_SIDE</code> | The target object is invoked within the current transaction whether there is a transaction or not. This policy depends on just-in-time transaction creation. |

You can enable just-in-time transactions by setting the following configuration variable to `true`:

```
plugins:ots:jit_transactions
```

If a transaction has begun but is not fully created, the transaction is created before the target object is invoked.

You cannot create a POA that mixes the `AUTOMATIC` or `SERVER_SIDE` [OTSPolicyValue](#) with the [EITHER](#) or [UNSHARED InvocationPolicyValue](#). Attempting to do this results in the [PortableServer::InvalidPolicy](#) exception being raised.

### See Also

[CosTransactions::OTSPolicyValue](#)





# IT\_CosTransactions::Current Class

This class extends the standard OTS `CosTransactions::Current` class with proprietary operations:

```
// Java
package com.iona.IT_CosTransactions;
public interface Current
    extends org.omg.CosTransactions.Current {

    public void commit_on_completion_of_next_call()
        throws org.omg.CosTransactions.NoTransaction
}

```

See Also [CosTransactions::Current](#)

## Current::commit\_on\_completion\_of\_next\_call()

This operation is used in conjunction with just-in-time transaction creation and the `SERVER_SIDE` `OTSPolicyValue`. This operation attempts to commit the current transaction immediately after the next invocation.

Using `commit_on_completion_of_next_call()` is logically equivalent to calling [Current::commit\(\)](#) immediately after the next invocation, except that the transaction is committed in the context of the target object. If there is no current transaction, a [NoTransaction](#) exception is raised.

---

**Note:** You should use this operation with caution.

---

See Also [CosTransactions::Current](#)  
[CosTransactions::Current::commit\(\)](#)  
[IT\\_CosTransactions::SERVER\\_SIDE](#)



# IT\_CSI Overview

The `IT_CSI` module defines Orbix-specific policy interfaces that enable you to set CSlv2 policies programmatically. An

`IT_CSI::IT_CSI_AUTH_METHOD_USERNAME_PASSWORD` constant is defined that enables you to create credentials on the client side using the `SecurityLevel2::PrincipalAuthenticator`. The module also defines proprietary credentials interfaces (giving you access to CSlv2-related credentials on the server side) and an `AuthenticateGSSUPCredentials` interface that enables you to implement a custom authentication service.

The module contains the following IDL interfaces:

- `IT_CSI::AttributeServicePolicy` Interface
- `IT_CSI::AuthenticationServicePolicy` Interface
- `IT_CSI::AttributeServiceProtocolClientPolicy` Interface
- `IT_CSI::AttributeServiceProtocolServerPolicy` Interface
- `IT_CSI::AuthenticateGSSUPCredentials` Interface
- `IT_CSI::CSICredentials` Interface
- `IT_CSI::CSIReceivedCredentials` Interface
- `IT_CSI::CSICurrent` Interface
- `IT_CSI::CSICurrent2` Interface
- `IT_CSI::CSICurrent3` Interface

Associated with the CSlv2 policies, the `IT_CSI` module defines the following policy type constants (of `CORBA::PolicyType` type):

```
IT_CSI::CSI_CLIENT_AS_POLICY
IT_CSI::CSI_SERVER_AS_POLICY
IT_CSI::CSI_CLIENT_SAS_POLICY
IT_CSI::CSI_SERVER_SAS_POLICY
IT_CSI::CSI_ATTRIBUTE_SERVICE_PROTOCOL_CLIENT_POLICY
IT_CSI::CSI_ATTRIBUTE_SERVICE_PROTOCOL_SERVER_POLICY
```

---

## IT\_CSI::IT\_CSI\_AUTH\_METH\_USERNAME\_PASSWORD

```
const Security::AuthenticationMethod
    IT_CSI_AUTH_METH_USERNAME_PASSWORD = 6;
```

This constant identifies CSiv2 username/password authentication method. When calling the

`SecurityLevel2::PrincipalAuthenticator::authenticate()` operation, the `IT_CSI_AUTH_METH_USERNAME_PASSWORD` constant can be passed as the method parameter.

In Java, to create a CSiv2 credentials object call the principal authenticator's `authenticate()` operation with its parameters set as follows:

```
//Java
...
authentication_result = principal_authenticator.authenticate(
    com.iona.IT_CSI.IT_CSI_AUTH_METH_USERNAME_PASSWORD.value,
    "", // NOT USED
    "<user_name>", // GSSUP user name
    auth_data_any, // an any containing an
                    // IT_CSI::GSSUPAuthData structure
    privileges, // NOT USED
    credentials, // returns the CSiv2 user credentials
    continuation_data, // NOT USED
    auth_specific_data // NOT USED
);
```

### See Also

```
SecurityLevel2::PrincipalAuthenticator
IT_CSI::GSSUPAuthData
```

## IT\_CSI::GSSUPAuthData Structure

```
struct GSSUPAuthData
{
    string password;
    string domain;
};
```

This structure is used to pass the GSSUP password and authentication domain name to the `SecurityLevel2::PrincipalAuthenticator::authenticate()` operation. It is used in combination with the

---

IT\_CSI::IT\_CSI\_AUTH\_METH\_USERNAME\_PASSWORD authentication method identifier.

In Java, an IT\_CSI::GSSUPAuthData structure must be inserted into an any before being passed as the auth\_data parameter of the SecurityLevel2::PrincipalAuthenticator::authenticate() operation. The IT\_CSI::GSSUPAuthData structure can be inserted into an any as follows:

```
//Java
org.omg.CORBA.Any auth_data_any =
    org.omg.CORBA.ORB.init().create_any();
com.iona.IT_CSI.GSSUPAuthData auth_data =
    new com.iona.IT_CSI.GSSUPAuthData("<password>", "<domain>");

// Insert the GSSUPAuthData struct into the any.
com.iona.IT_CSI.GSSUPAuthDataHelper.insert(
    auth_data_any,
    auth_data
);
```

This structure contains the following fields:

password	The GSSUP password for this login.
domain	The CSIv2 authentication domain for this login.

**See Also**

IT\_CSI::IT\_CSI\_AUTH\_METH\_USERNAME\_PASSWORD

## IT\_CSI::CSI\_POLICY\_BASE

```
const unsigned long CSI_POLICY_BASE =
    IT_PolicyBase::IONA_POLICY_ID + 11;
```

The base for a range of CSIv2 policy constants.

**See Also**

```
IT_CSI::CSI_CLIENT_AS_POLICY
IT_CSI::CSI_SERVER_AS_POLICY
IT_CSI::CSI_CLIENT_SAS_POLICY
IT_CSI::CSI_SERVER_SAS_POLICY
```

---

## IT\_CSI::CSI\_CLIENT\_AS\_POLICY

```
const CORBA::PolicyType CSI_CLIENT_AS_POLICY = CSI_POLICY_BASE;
```

*(Deprecated)* The flag identifying the client-side authentication service policy.

### See Also

IT\_CSI::CSI\_SERVER\_AS\_POLICY

IT\_CSI::AuthenticationServicePolicy

## IT\_CSI::CSI\_SERVER\_AS\_POLICY

```
const CORBA::PolicyType CSI_SERVER_AS_POLICY = CSI_POLICY_BASE+1;
```

*(Deprecated)* The flag identifying the server-side authentication service policy.

### See Also

IT\_CSI::CSI\_CLIENT\_AS\_POLICY

IT\_CSI::AuthenticationServicePolicy

## IT\_CSI::CSI\_CLIENT\_SAS\_POLICY

```
const CORBA::PolicyType CSI_CLIENT_SAS_POLICY = CSI_POLICY_BASE+2;
```

*(Deprecated)* The flag identifying the client-side attribute service policy.

### See Also

IT\_CSI::CSI\_SERVER\_SAS\_POLICY

IT\_CSI::AttributeServicePolicy

## IT\_CSI::CSI\_SERVER\_SAS\_POLICY

```
const CORBA::PolicyType CSI_SERVER_SAS_POLICY = CSI_POLICY_BASE+3;
```

*(Deprecated)* The flag identifying the server-side attribute service policy.

### See Also

IT\_CSI::CSI\_CLIENT\_SAS\_POLICY

IT\_CSI::AttributeServicePolicy

---

## IT\_CSI::CSI\_ATTRIBUTE\_SERVICE\_PROTOCOL\_CLIENT\_POLICY

```
const CORBA::PolicyType  
CSI_ATTRIBUTE_SERVICE_PROTOCOL_CLIENT_POLICY = 0x49545F97;
```

The flag identifying the policy for client-side CSv2 settings (used in combination with the `IT_CSI::AttributeServiceProtocolClientPolicy` policy).

## IT\_CSI::CSI\_ATTRIBUTE\_SERVICE\_PROTOCOL\_SERVER\_POLICY

```
const CORBA::PolicyType  
CSI_ATTRIBUTE_SERVICE_PROTOCOL_SERVER_POLICY = 0x49545F98
```

The flag identifying the policy for server-side CSv2 settings (used in combination with the `IT_CSI::AttributeServiceProtocolServerPolicy` policy).

## IT\_CSI::AuthenticationService Structure

```
struct AuthenticationService  
{  
    // Client and server side.  
    CSIIOP::AssociationOptions support;  
  
    // Server side only.  
    CSIIOP::AssociationOptions requires;  
    string client_authentication_mech;  
    string target_name;  
    AuthenticateGSSUPCredentials as_object;  
};
```

*(Deprecated)* This structure, in conjunction with the `IT_CSI::AuthenticationServicePolicy` interface, provides a programmatic approach to enabling the CSv2 authentication service policy. This structure has a dual purpose, because it can be used to set both a client-side policy, `IT_CSI::CSI_CLIENT_AS_POLICY`, and a server-side policy, `IT_CSI::CSI_SERVER_AS_POLICY`.

---

This structure contains the following fields:

<code>support</code>	<i>(Client and server)</i> The list of association options <i>supported</i> by the authentication service policy. Currently, only the <code>CSIIOP::EstablishTrustInClient</code> association option can be included in this list.
<code>requires</code>	<i>(Server only)</i> The list of association options <i>required</i> by the authentication service policy on the server side. Currently, only the <code>CSIIOP::EstablishTrustInClient</code> association option can be included in this list.
<code>client_authentication_mech</code>	<i>(Server only)</i> The authentication mechanism OID, which identifies the mechanism used by CSIV2 authentication. For example, <code>GSSUP::GSSUPMechOID</code> is a valid setting.
<code>target_name</code>	<i>(Server only)</i> The name of the security policy domain (CSIV2 authentication domain) for this authentication service.
<code>as_object</code>	<i>(Server only)</i> A reference to the GSSUP authentication service object that will be used to authenticate GSS username/password combinations on the server side.

**See Also**

`IT_CSI::AuthenticationServicePolicy`  
`IT_CSI::CSI_CLIENT_AS_POLICY`  
`IT_CSI::CSI_SERVER_AS_POLICY`

## **IT\_CSI::SupportedNamingMechanisms Sequence**

```
typedef sequence<string> SupportedNamingMechanisms;
```

*(Deprecated)* The list of naming mechanisms supported by CSIV2. Currently, the only supported naming mechanism is `CSI::GSS_NT_Export_Name_OID`.

**See Also**

`CSI::GSS_NT_Export_Name_OID`  
`IT_CSI::AttributeService`



---

## IT\_CSI::AttributeService Structure

```
struct AttributeService
{
    CSIIOP::AssociationOptions support;
    SupportedNamingMechanisms supported_naming_mechs;
    CSI::IdentityTokenType supported_identity_types;
};
```

*(Deprecated)* This structure, in conjunction with the `IT_CSI::AttributeServicePolicy` interface, provides a programmatic approach to enabling the CSIv2 attribute service policy. This structure has a dual purpose, because it can be used to set both a client-side policy, `IT_CSI::CSI_CLIENT_SAS_POLICY`, and a server-side policy, `IT_CSI::CSI_SERVER_SAS_POLICY`.

This structure contains the following fields:

<code>support</code>	<i>(Client and server)</i> The list of association options <i>supported</i> by the attribute service policy. Currently, only the <code>CSIIOP::IdentityAssertion</code> association option can be included in this list.
<code>supported_naming_mechs</code>	<i>(Server only)</i> A list of GSS naming mechanism OIDs, which identify the formats that may be used in the <code>CSI::ITTPrincipalName</code> identity token. For example, <code>CSI::GSS_NT_Export_Name_OID</code> is a valid naming mechanism string.
<code>supported_identity_types</code>	<i>(Server only)</i> The bitmapped representation of the set of identity token types supported by the target. In the current implementation of Orbix, the value of this attribute should be <code>0x03</code> (which represents a combination of the <code>ITTAnonymous</code> flag and the <code>ITTPrincipalName</code> flag)..

### See Also

```
IT_CSI::AttributeServicePolicy
CSI::GSS_NT_Export_Name_OID
IT_CSI::CSI_CLIENT_SAS_POLICY
IT_CSI::CSI_SERVER_SAS_POLICY
```

---

## IT\_CSI::AttributeServiceProtocolClient

```
struct AttributeServiceProtocolClient
{
    CSIIOP::AssociationOptions auth_over_trans_supports;
    CSIIOP::AssociationOptions attribute_service_layer_supports;
};
```

This structure, in conjunction with the `IT_CSI::AttributeServiceProtocolClientPolicy` interface, provides a programmatic approach to enabling the *client side* of the CSv2 authentication over transport mechanism and the CSv2 identity assertion mechanism. To create a policy of `IT_CSI::AttributeServiceProtocolClientPolicy` type, insert an `IT_CSI::AttributeServiceProtocolClient` struct into a CORBA any and call the `CORBA::ORB::create_policy()` operation with `IT_CSI::ATTRIBUTE_SERVICE_PROTOCOL_SERVICE_CLIENT_POLICY` as the first argument.

This structure contains the following fields:

<code>auth_over_trans_supports</code>	The list of association options <i>supported</i> by the authentication over transport layer on the client side. Currently, only the <code>CSIIOP::EstablishTrustInClient</code> association option can be included in this list.
<code>attribute_service_layer_supports</code>	The list of association options <i>supported</i> by the attribute service layer on the client side. Currently, only the <code>CSIIOP::IdentityAssertion</code> association option can be included in this list.

## IT\_CSI::AttributeServiceProtocolServer

```
struct AttributeServiceProtocolServer
{
    string server_domain_name;
    CSIIOP::AssociationOptions auth_over_trans_supports;
    CSIIOP::AssociationOptions auth_over_trans_requires;
    CSIIOP::AssociationOptions attribute_service_layer_supports;
};
```

---

This structure, in conjunction with the

`IT_CSI::AttributeServiceProtocolServerPolicy` interface, provides a programmatic approach to enabling the *server side* of the CSiv2 authentication over transport mechanism and the CSiv2 identity assertion mechanism. To create a policy of `IT_CSI::AttributeServiceProtocolServerPolicy` type, insert an `IT_CSI::AttributeServiceProtocolServer` struct into a CORBA any and call the `CORBA::ORB::create_policy()` operation with `IT_CSI::ATTRIBUTE_SERVICE_PROTOCOL_SERVICE_SERVER_POLICY` as the first argument.

This structure contains the following fields:

<code>server_domain_name</code>	The name of the security policy domain (CSiv2 authentication domain) for this authentication service.
<code>auth_over_trans_supports</code>	The list of association options <i>supported</i> by the authentication over transport layer on the server side. Currently, only the <code>CSIIOP::EstablishTrustInClient</code> association option can be included in this list.
<code>auth_over_trans_requires</code>	The list of association options <i>required</i> by the authentication over transport layer on the server side. Currently, only the <code>CSIIOP::EstablishTrustInClient</code> association option can be included in this list.
<code>attribute_service_layer_supports</code>	The list of association options <i>supported</i> by the attribute service layer on the server side. Currently, only the <code>CSIIOP::IdentityAssertion</code> association option can be included in this list.

## IT\_CSI::CSICredentialsType Enumeration

```
enum CSICredentialsType {
    GSSUPCredentials,
    PropagatedCredentials,
    TransportCredentials
};
```

---

An enumeration to identify the type of credentials contained in a CSiv2 credentials object. The credentials can be one of the following types:

- `GSSUPCredentials`—a set of GSS username/password credentials (authenticated on the server side), received through the CSiv2 authorization over transport mechanism.
- `PropagatedCredentials`—a set of propagated credentials (not authenticated on the server side), received through the CSiv2 identity assertion mechanism.
- `TransportCredentials`—a set of SSL/TLS credentials (typically containing an X.509 certificate chain), received through the transport layer.

**See Also**

`IT_CSI::CSICredentials`

`IT_CSI::CSIReceivedCredentials`

# IT\_CSI::AttributeServicePolicy Interface

```
// IDL in module IT_CSI
local interface AttributeServicePolicy : CORBA::Policy
{
    // The following attribute, supports, is for client and server
    // side
    readonly attribute CSIIOP::AssociationOptions support;

    // Server specific attributes used in IOR generation
    readonly attribute CSI::OIDList supported_naming_mechanisms;
    readonly attribute CSI::IdentityTokenType
        supported_identity_types;
    readonly attribute boolean backward_trust_enabled;
    readonly attribute CSIIOP::ServiceConfigurationList
        privilege_authorities;
};
```

*(Deprecated)* The policy type for the CSv2 attribute service policy, which is used to enable the CSv2 *identity assertion* mechanism. This interface, in conjunction with the `IT_CSI::AttributeService` struct, provides a programmatic approach to enabling the CSv2 attribute service policy. The functionality provided is equivalent to that which is available by setting the following configuration variables:

```
policies:csi:attribute_service:client_supports
policies:csi:attribute_service:target_supports
policies:csi:attribute_service:backward_trust:enabled
```

This `AttributeServicePolicy` interface has a dual purpose. It can represent either a client-side policy, `IT_CSI::CSI_CLIENT_SAS_POLICY`, or a server-side policy, `IT_CSI::CSI_SERVER_SAS_POLICY`.

---

**Note:** Instead of using this policy, you can optionally use the `AttributeServiceProtocolClientPolicy` and `AttributeServiceProtocolServerPolicy` policies to the identity assertion mechanism. These alternative policies require you to set `plugins:csi:use_legacy_policies` to `false` in the Orbix configuration.

---

## See Also

`IT_CSI::CSI_CLIENT_SAS_POLICY`  
`IT_CSI::CSI_SERVER_SAS_POLICY`  
`IT_CSI::AttributeService`  
`IT_CSI::AuthenticationServicePolicy`

## AttributeServicePolicy::support

`readonly attribute CSIIOP::AssociationOptions support;`

*(Deprecated)* The list of association options *supported* by the attribute service policy. Currently, only the `CSIIOP::IdentityAssertion` association option can be included in this list.

The effect of including the `CSIIOP::IdentityAssertion` association option in the list depends on whether the `AttributeServicePolicy` is set as a client-side policy (`IT_CSI::CSI_CLIENT_SAS_POLICY`) or as a server-side policy (`IT_CSI::CSI_SERVER_SAS_POLICY`), as follows:

- Client side—supports the propagation of an identity to the server using the CSv2 identity assertion mechanism. This is equivalent to the `policies:csi:attribute_service:client_supports` configuration variable.
- Server side—supports the receipt of an identity (which is presumed to have been already authenticated) from the client using the CSv2 identity assertion mechanism. This is equivalent to the `policies:csi:attribute_service:target_supports` configuration variable.

## See Also

`CSIIOP::IdentityAssertion`

---

## AttributeServicePolicy::supported\_naming\_mechanisms

readonly attribute CSI::OIDList supported\_naming\_mechanisms;

*(Deprecated)* A list of GSS naming mechanism OIDs, which identify the formats that may be used in the CSI::ITTPrincipalName identity token. In the current implementation of Orbix, the supported\_naming\_mechanisms list would normally include a binary representation of the CSI::GSS\_NT\_Export\_Name\_OID naming mechanism OID.

### See Also

CSI::ITTPrincipalName  
CSI::GSS\_NT\_Export\_Name\_OID

## AttributeServicePolicy::supported\_identity\_types

readonly attribute CSI::IdentityTokenType  
supported\_identity\_types;

*(Deprecated)* The bitmapped representation of the set of identity token types supported by the target. In the current implementation of Orbix, the value of this attribute would be 0x00000003, which represents a combination of the ITTAnonymous flag (0x01) and the ITTPrincipalName flag (0x02). The ITTAbsent identity token is always supported.

The ITTX509CertChain identity token and the ITTDistinguishedName identity token are *not* supported in the current implementation. Hence, the corresponding flags for these identity tokens cannot be set.

### See Also

CSI::ITTAbsent  
CSI::ITTAnonymous  
CSI::ITTPrincipalName

## AttributeServicePolicy::backward\_trust\_enabled

readonly attribute boolean backward\_trust\_enabled;

*(Deprecated)* Not used in the current implementation.

---

## **AttributeServicePolicy::privilege\_authorities**

```
readonly attribute CSIIOP::ServiceConfigurationList  
    privilege_authorities;
```

*(Deprecated)* A list of authorization tokens. This feature is currently not supported by Orbix (that is, it returns an empty list).



# IT\_CSI::AttributeServiceProtocolClientPolicy Interface

```
// IDL in module IT_CSI
local interface AttributeServiceProtocolClientPolicy
    : CORBA::Policy
{
    readonly attribute AttributeServiceProtocolClient
        attribute_service_protocol_client;
};
```

The policy type for configuring CSiv2 mechanisms on the client side. Two mechanisms for transmitting credentials are provided by CSiv2:

- *Authentication over transport mechanism*—for transmitting username and password credentials.
- *Attribute service layer mechanism*—for transmitting an asserted identity.

This interface, in conjunction with the `IT_CSI::AttributeServiceProtocolClient` struct, provides a programmatic approach to enabling the CSiv2 mechanisms on the client side. The functionality provided is equivalent to that which is available by setting the following configuration variables:

```
policies:csi:auth_over_transport:client_supports
policies:csi:attribute_service:client_supports
```

When creating a policy of this type using the `CORBA::ORB::create_policy()` operation, you should pass in the `IT_CSI::CSI_ATTRIBUTE_SERVICE_PROTOCOL_CLIENT_POLICY` policy constant as the first argument.

---

**Note:** In order to use this interface, you must set the `plugins:csi:use_legacy_policies` configuration variable to `false` (that is, this interface replaces the legacy `IT_CSI::AuthenticationServicePolicy` and `IT_CSI::AttributeServicePolicy` interfaces).

---

### **AttributeServiceProtocolClientPolicy::attribute\_service\_protocol\_client Attribute**

```
readonly attribute AttributeServiceProtocolClient
    attribute_service_protocol_client;
```

Returns an instance of an `IT_CSI::AttributeServiceProtocolClient` struct, which contains the client-side association options for the CSlv2 authentication over transport mechanism and the CSlv2 attribute service layer mechanism.

# IT\_CSI::AttributeServiceProtocolServerPolicy Interface

```
// IDL in module IT_CSI
local interface AttributeServiceProtocolServerPolicy
: CORBA::Policy
{
    readonly attribute AttributeServiceProtocolServer
        attribute_service_protocol_server;
};
```

The policy type for configuring CSiv2 mechanisms on the server side. Two mechanisms for transmitting credentials are provided by CSiv2:

- *Authentication over transport mechanism*—for transmitting username and password credentials.
- *Attribute service layer mechanism*—for transmitting an asserted identity.

This interface, in conjunction with the

`IT_CSI::AttributeServiceProtocolServer` struct, provides a programmatic approach to enabling the CSiv2 mechanisms on the server side. The functionality provided is equivalent to that which is available by setting the following configuration variables:

```
policies:csi:auth_over_transport:target_supports
policies:csi:auth_over_transport:target_requires
policies:csi:auth_over_transport:server_domain_name
policies:csi:attribute_service:target_supports
```

When creating a policy of this type using the `CORBA::ORB::create_policy()` operation, you should pass in the

`IT_CSI::CSI_ATTRIBUTE_SERVICE_PROTOCOL_SERVER_POLICY` policy constant as the first argument.

---

**Note:** In order to use this interface, you must set the `plugins:csi:use_legacy_policies` configuration variable to `false` (that is, this interface replaces the legacy `IT_CSI::AuthenticationServicePolicy` and `IT_CSI::AttributeServicePolicy` interfaces).

---

### **AttributeServiceProtocolServerPolicy::attribute\_service\_protocol\_server Attribute**

```
readonly attribute AttributeServiceProtocolServer
    attribute_service_protocol_server;
```

Returns an instance of an `IT_CSI::AttributeServiceProtocolServer` struct, which contains the server-side association options for the CSlv2 authentication over transport mechanism and the CSlv2 attribute service layer mechanism.

# IT\_CSI::AuthenticateGSSUPCredentials Interface

```
// IDL in module IT_CSI
interface AuthenticateGSSUPCredentials
{
    readonly attribute string authentication_service;

    boolean authenticate (
        in string username,
        in string password,
        in string target_name,
        out GSSUP::ErrorCode error_code);
};
```

A callback interface that you can optionally implement to provide a custom authentication service for a CSiv2 server. When using the CSiv2 authentication over transport mechanism (enabled by the CSiv2 authentication service policy), the `AuthenticateGSSUPCredentials::authenticate()` operation is invoked for every incoming request from a client. This gives you the opportunity to accept or reject every incoming invocation based on the authentication data provided by the client.

Note that this *stateless* mode of operation (calling `authenticate()` for every invocation) is the only kind of session semantics currently supported by Orbix. The *stateful* mode of operation (calling `authenticate()` once at the beginning of a session) is currently *not* supported.

You can install an implementation of `AuthenticateGSSUPCredentials` in one of the following ways:

- *By configuration*—you can specify the `AuthenticateGSSUPCredentials` implementation class by setting the following configuration variable:  
`policies:csi:auth_over_transport:authentication_service`  
The named class is then loaded and instantiated by the CSiv2 plug-in.

- 
- *By programming*—you can register an instance of the `AuthenticateGSSUPCredentials` implementation class by setting the `as_object` field of the `IT_CSI::AuthenticationServicePolicy`.
  - *By setting an initial reference*—you can register an instance of the `AuthenticateGSSUPCredentials` implementation using the initial reference mechanism. Use the initial reference ID, `IT_CSIAuthenticationObject`, to register the instance.

## Examples

The following Java example defines the `AuthenticateGSSUPImpl` class, which is a sample implementation of the `AuthenticateGSSUPCredentials` object. This sample implementation rejects all requests except for those from `username` user, with `password` password, belonging to `PC-DOMAIN` security policy domain, and invoking the `call_me()` operation.

```
// Java
package demos.csiv2.auth_service;

import com.iona.corba.IT_CSI.*;
import org.omg.CORBA.IntHolder;

public class AuthenticateGSSUPImpl
extends org.omg.CORBA.LocalObject
implements AuthenticateGSSUPCredentials
{
    public String authentication_service()
    {
        return "Demo Authentication Service";
    }

    public boolean authenticate(String username,
                               String password,
                               String target_name,
                               String request_name,
                               String object_name,
                               org.omg.CORBA.IntHolder error_code)
    {
        boolean authentication_result = false;

        System.out.println("Username received " + username);
        System.out.println("password received " + password);
        System.out.println("target_name received " + target_name);
        System.out.println("request_name received " + request_name);
        System.out.println("object_name received " + object_name);
    }
}
```

---

```

        if ( username.equals("username"))
        {
            if ( password.equals("password"))
            {
                if ( target_name.equals("PC-DOMAIN"))
                {
                    if ( request_name.equals("call_me"))
                    {
                        System.out.println("Accepting request " +
request_name + " for above");
                        authentication_result = true;
                    }
                    else
                    {
                        error_code.value =
org.omg.GSSUP.GSS_UP_S_G_UNSPECIFIED.value;
                    }
                }
                else
                {
                    error_code.value =
org.omg.GSSUP.GSS_UP_S_G_BAD_TARGET.value;
                }
            }
            else
            {
                error_code.value =
org.omg.GSSUP.GSS_UP_S_G_BAD_PASSWORD.value;
            }
        }
        else
        {
            error_code.value = org.omg.GSSUP.GSS_UP_S_G_NOUSER.value;
        }

        return authentication_result;
    }
}

```

**See Also**

IT\_CSI::AuthenticationServicePolicy

---

## AuthenticateGSSUPCredentials::authentication\_service Attribute

```
readonly attribute string authentication_service;
```

The name of the authentication service implementation. There are no particular conditions imposed on the value of this attribute; it is just a short descriptive string.

## AuthenticateGSSUPCredentials::authenticate()

```
boolean authenticate (  
    in string username,  
    in string password,  
    in string target_name,  
    in string request_name,  
    in string object_name,  
    out GSSUP::ErrorCode error_code);
```

A callback operation that performs authentication on a GSSUP username/password combination. When CSIV2 authentication over transport is enabled, the `authenticate()` operation is called for every incoming request on the server side. If the return value is `TRUE`, the request is allowed to proceed; if the return value is `FALSE`, the request is rejected.

### Parameters

The `authenticate()` operation takes the following parameters:

<code>username</code>	The username received from the client through the CSIV2 authentication over transport mechanism.
<code>password</code>	The password received from the client through the CSIV2 authentication over transport mechanism.
<code>target_name</code>	The security policy domain name (CSIV2 authentication domain) received from the client through the CSIV2 authentication over transport mechanism.
<code>request_name</code>	The name of the operation (or attribute accessor/modifier) to be invoked by this request. The format of this argument is the same as the operation name in a GIOP request header. See, for example, the description of <code>GIOP::RequestHeader_1_2::operation</code> in section 15.4.2 of the CORBA 2.4.2 core specification.



---

<code>object_name</code>	The type identifier for the target of this invocation, expressed as a CORBA repository ID. For example, the <code>CosNaming::NamingContext</code> type would be identified by the IDL: <code>omg.org/CosNaming/NamingContext:1.0</code> repository ID string.
<code>error_code</code>	The returned GSSUP error code (long integer). A non-zero value indicates that an error occurred.

**See Also**

`IT_CSI::AuthenticationServicePolicy`



# IT\_CSI::AuthenticationServicePolicy Interface

```
// IDL in module IT_CSI
local interface AuthenticationServicePolicy : CORBA::Policy
{
    // The following attribute, supports, is for client and server
    // side
    readonly attribute CSIIOP::AssociationOptions support;

    // Server specific attributes used in IOR generation
    readonly attribute CSIIOP::AssociationOptions target_requires;
    readonly attribute CSI::OID client_authentication_mech;
    readonly attribute CSI::GSS_NT_ExportedName target_name;
    readonly attribute AuthenticateGSSUPCredentials as_object;
};
```

*(Deprecated)* The policy type for the CSv2 authentication service policy, which is used to enable the CSv2 *authentication over transport* mechanism. This interface, in conjunction with the `IT_CSI::AuthenticationService` struct, provides a programmatic approach to enabling the CSv2 authentication service policy. The functionality provided is equivalent to that which is available by setting the following configuration variables:

```
policies:csi:auth_over_transport:client_supports
policies:csi:auth_over_transport:target_supports
policies:csi:auth_over_transport:target_requires
policies:csi:auth_over_transport:server_domain_name
policies:csi:auth_over_transport:authentication_service
```

This `AuthenticationServicePolicy` interface has a dual purpose. It can represent either a client-side authentication policy, `IT_CSI::CSI_CLIENT_AS_POLICY`, or a server-side authentication policy, `IT_CSI::CSI_SERVER_AS_POLICY`.

---

**Note:** Instead of using this policy, you can optionally use the `AttributeServiceProtocolClientPolicy` and `AttributeServiceProtocolServerPolicy` policies to control authorization over transport. These alternative policies require you to set `plugins:csi:use_legacy_policies` to `false` in the Orbix configuration.

---

## Examples

The following Java example shows how to set the authentication service policy at the ORB level:

```
// Java
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CSIIOP.EstablishTrustInClient;
import org.omg.GSSUP.GSSUPMechOID;

import org.omg.Security.*;
import org.omg.SecurityLevel2.*;

import com.ionacorba.IT_CSI.CSI_SERVER_AS_POLICY;
...
try
{
    // Note the following:
    //
    // m_orb - an ORB instance, already initialized.
    //
    // AuthenticateGSSUPImpl - an implementation of
    //     org.omg.corba.IT_CSI.AuthenticateGSSUPCredentials
    //
    AuthenticateGSSUPImpl as_obj = new AuthenticateGSSUPImpl();

    com.ionacorba.IT_CSI.AuthenticationService as =
        new com.ionacorba.IT_CSI.AuthenticationService(
            EstablishTrustInClient.value,
            EstablishTrustInClient.value,
            GSSUPMechOID.value,
            "PCGROUP", // an authentication domain
            as_obj
        );

    Any any = m_orb.create_any();
```

---

```

com.ionacorba.IT_CSI.AuthenticationServiceHelper.insert(
    any,
    as
);

PolicyManager pol_manager=
(PolicyManager)m_orb.resolve_initial_references(
    "ORBPolicyManager"
);

Policy[] policies = new Policy[1];
policies[0] = m_orb.create_policy(
    CSI_SERVER_AS_POLICY.value,
    any
);

pol_manager.set_policy_overrides(
    policies,
    SetOverrideType.SET_OVERRIDE
);
}
catch (java.lang.Exception ex) { /* Handle exceptions */ }

```

## AuthenticationServicePolicy::support Attribute

readonly attribute CSIIOP::AssociationOptions support;

*(Deprecated)* The list of association options *supported* by the authentication service policy. Currently, only the CSIIOP::EstablishTrustInClient association option can be included in this list.

The CSIIOP::EstablishTrustInClient association option can be set either as a client-side policy (IT\_CSI::CSI\_CLIENT\_AS\_POLICY) or as a server-side policy (IT\_CSI::CSI\_SERVER\_AS\_POLICY), as follows:

- Client side—supports the propagation of a GSSUP username and password using the CSv2 authentication mechanism. This is equivalent to the `policies:csi:auth_over_transport:client_supports` configuration variable.
- Server side—supports the authentication of a client's username and password using the CSv2 authentication mechanism. This is equivalent to the `policies:csi:auth_over_transport:target_supports` configuration variable.

---

## AuthenticationServicePolicy::target\_requires Attribute

readonly attribute CSIIOP::AssociationOptions target\_requires;

*(Deprecated)* The list of association options *required* by the authentication service policy on the server side. Currently, only the CSIIOP::EstablishTrustInClient association option can be included in this list.

## AuthenticationServicePolicy::client\_authentication\_mech Attribute

readonly attribute CSI::OID client\_authentication\_mech;

*(Deprecated)* The authentication mechanism OID, which identifies the mechanism used by CSIV2 authentication on the server side. In the current implementation of Orbix, the only available mechanism is the Generic Security Service username/password (GSSUP) mechanism, represented by GSSUP::GSSUPMechOID.

### See Also

GSSUP::GSSUPMechOID  
CSI::StringOID

## AuthenticationServicePolicy::target\_name Attribute

readonly attribute CSI::GSS\_NT\_ExportedName target\_name;

*(Deprecated)* The name of the security policy domain (CSIV2 authentication domain) for this authentication service on the server side. In this implementation, a given CSIV2 server can belong to a single security policy domain only. If an incoming client request does not match the server's security policy domain, the client request will be rejected.

---

## **AuthenticationServicePolicy::as\_object Attribute**

readonly attribute AuthenticateGSSUPCredentials as\_object;

*(Deprecated)* A reference to the GSSUP authentication service object that will be used to authenticate GSS username/password combinations on the server side.





# IT\_CSI::CredentialsListObject Interface

```
// IDL in module IT_CSI
local interface CredentialsListObject
{
    // Efficient non copy variant of above, the return type is NOT
    // a deep copy
    // and can only be used while the owning CredentialsListObject
    // is alive.
    SecurityLevel2::CredentialsList credentials_list_nc();
};
```

Interface that serves as a wrapper around a `CredentialsList` sequence and is used to avoid the cost associated with returning sequences as deep copies. This interface type is used by the `IT_CSI::CSICurrent3` interface.

## **CredentialsListObject::credentials\_list\_nc()**

```
SecurityLevel2::CredentialsList credentials_list_nc();
```

Returns a pointer to the contained `CredentialsList` sequence, *without* making a deep copy; the returned sequence can only be used so long as the containing `CredentialsListObject` remains alive.

### **See Also**

```
SecurityLevel2::Credentials
Security::SecAttribute
```



# IT\_CSI::CSICredentials Interface

```
local interface CSICredentials : SecurityLevel2::Credentials
{
    readonly attribute CSICredentialsType csi_credentials_type;
};
```

IONA-specific `CSICredentials` interface that is used as a base interface for CSlv2 credentials. Server implementations may use this interface to determine the clients credentials type—for example, a propagated identity from an intermediary or a username/password.

## **CSICredentials::csi\_credentials\_type Attribute**

```
readonly attribute CSICredentialsType csi_credentials_type;
```

A flag that indicates what type of credentials is returned by the `SecurityLevel2::Current::received_credentials()` operation.

### **See Also**

`IT_CSI::CSIReceivedCredentials`



# IT\_CSI::CSICurrent Interface

```
// IDL in module IT_CSI
local interface CSICurrent : CORBA::Current
{
    boolean set_received_gssup_credentials(in string access_id);
};
```

The operations in this interface are now *deprecated*. Use the IT\_CSI::CSICurrent2 interface instead.

## CSICurrent::set\_received\_gssup\_credentials()

```
boolean set_received_gssup_credentials(in string access_id);
```

Deprecated. Use

```
IT_CSI::CSICurrent2::set_received_gssup_credentials_access_id()
instead.
```

### Parameters

This operation takes the following parameters:

<code>access_id</code>	Either the GSSUP username in string format or the common name from an X.509 certificate's subject DN. From the target server, the access ID is made accessible from a <code>Security::SecAttribute::value</code> in the form of an <code>AccessId</code> encoded as a sequence of octets.
------------------------	---

### See Also

```
SecurityLevel2::ReceivedCredentials
SecurityLevel2::Credentials
Security::SecAttribute
```



# IT\_CSI::CSICurrent2 Interface

```
// IDL in module IT_CSI
local interface CSICurrent2 : CSICurrent
{
    CSIReceivedCredentials
    set_received_gssup_credentials_access_id(
        in string peer_identity
    );

    CSIReceivedCredentials
    set_received_itt_principal_name_identity_token(
        in string asserted_identity
    );

    boolean
    set_csi_received_credentials(
        in CSIReceivedCredentials          rec_creds
    );
};
```

Interface used to set the value of the CSI received credentials in the current execution context. By calling the operations in this interface, you can *simulate* the successfully processed receipt of a CSIV2 asserted identity message and/or the receipt and successful processing of a CSIV2 GSSUP authentication request. These operations should be used only when you do not actually have a CSIV2 execution context; for example, if you were building a bridge between the SOAP protocol and the CORBA GIOP protocol.

---

---

**WARNING:** It is critically important to understand that it is *your* responsibility to vet the user identities passed to the `CSICurrent2` operations. If you pass the identity of an unauthorized user into the CSI received credentials object, you could potentially undermine the security of your system completely.

---

---

---

A typical CSIV2 identity assertion scenario involves a client, an intermediate server, and a target server. The client invokes an operation on the intermediate server, with CSIV2 authentication over transport enabled, and the intermediate server invokes an operation on the target server, with CSIV2 identity assertion enabled.

Default values of the CSI received credentials are set automatically by parsing the appropriate GIOP service contexts from the incoming request message. In this case, it is recommended that you do *not* modify the CSI received credentials. The `CSICurrent2` interface is meant to be used *only* to simulate CSI received credentials in a bridging application, not to replace existing credentials.

A programmer can access an `IT_CSI::CSICurrent2` object from within an operation context using the following code:

```
// Java
com.ionacorba.IT_CSI.CSICurrent2 it_csi_current = null;
try {
    org.omg.CORBA.Object objref =
        orb.resolve_initial_references("SecurityCurrent");

    it_csi_current =
        com.ionacorba.IT_CSI.CSICurrent2Helper.narrow(objref);
}
catch (org.omg.CORBA.ORBPackage.InvalidName ex) {
    // Error: resolve_initial_references() call failed...
}
catch (org.omg.CORBA.BAD_PARAM ex) {
    // Error: narrow() call failed...
}
```

### **CSICurrent2::set\_received\_gssup\_credentials\_access\_id()**

```
CSIReceivedCredentials
set_received_gssup_credentials_access_id(
    in string peer_identity
);
```

Sets the GSSUP username attribute (or access ID, in the terminology of the OMG CORBASEC specification) for the peer identity in the CSI received credentials object, replacing whatever value was previously stored.



---

The main reason for calling this operation is to simulate the receipt of GSSUP credentials when bridging from a protocol that does not support the CSI authentication over transport mechanism. The next time the application invokes a remote operation within the current execution context, the CSI asserted identity used for the invocation is one of the following:

- The received identity token (set by the `set_received_itt_principal_name_identity_token()` operation), if present, otherwise
- The received GSSUP username (set by the `set_received_gssup_credentials_access_id()` operation), if present.

This operation replaces the deprecated

`IT_CSI::CSICurrent::set_received_gssup_credentials()` operation.

Returns a reference to the created or updated CSI received credentials object if the operation is successful; otherwise, returns a nil object reference.

---

**Note:** There is no option to set the password and domain name along with the GSSUP username. This is because the received GSSUP credentials are created *after* the GSSUP username has been authenticated. Hence, the password and domain name are not needed at this point and they are not stored in the received credentials.

---

## Parameters

This operation takes the following parameters:

`peer_identity` A GSSUP username to set or replace the value stored in the CSI received credentials. If present, the original stored value would have been parsed from the incoming request message.

## See Also

`SecurityLevel2::ReceivedCredentials`  
`SecurityLevel2::Credentials`  
`Security::SecAttribute`

## `CSICurrent2::set_received_itt_principal_name_identity_token()`

`CSIReceivedCredentials`  
`set_received_itt_principal_name_identity_token(`

---

```
        in string asserted_identity
    );
```

Sets the CSI asserted identity in the CSI received credentials object, replacing whatever value was previously stored and implicitly setting the identity token type to be `ITTPPrincipalName`.

The main reason for calling this operation is to simulate the receipt of a CSI identity token when bridging from a protocol that does not support the CSI identity assertion mechanism. The next time the application invokes a remote operation within the current execution context, the CSI identity assertion mechanism uses the identity token set by this operation.

Returns a reference to the created or updated CSI received credentials object if the operation is successful; otherwise, returns a nil object reference.

### Parameters

This operation takes the following parameters:

<code>asserted_identity</code>	An asserted identity to set or replace the value stored in the CSI received credentials. If present, the original stored value would have been parsed from the incoming request message.
--------------------------------	--

### **CSICurrent2::set\_csi\_received\_credentials()**

```
boolean
set_csi_received_credentials(
    in CSISReceivedCredentials    rec_creds
);
```

Initialize the CSI received credentials for the current execution context, replacing whatever value was previously stored in this context (if any).

The main reason for calling this operation would be to copy CSI received credentials between threads. For example, if a mid-tier server decides to spawn a new thread to make a follow-on invocation, you would need to copy the received credentials from the original thread (see `SecurityLevel2::Current::received_credentials()`) and call `set_csi_received_credentials()` in the newly-spawned thread, in order to ensure that the CSI credentials are propagated to the next tier.

---

Returns `true` if the CSI received credentials are successfully initialized; otherwise, `false`.

**Parameters** This operation takes the following parameters:

`rec_creds` A received credentials object to set or replace the value stored in the current CSI received credentials.



# IT\_CSI::CSICurrent3 Interface

```
// IDL in module IT_CSI
local interface CSICurrent3 : CSICurrent2
{
    CredentialsListObject
    set_effective_own_gssup_credentials_info(
        in string    username,
        in string    password,
        in string    domain
    );

    void
    set_effective_own_credentials_list(
        in CredentialsListObject    effective_own_credentials
    );
};
```

Interface used to set the value of the CSI GSSUP own credentials in the current execution context. There are, in fact, two different kinds of credentials supported by the CSI protocol, as follows:

- *GSSUP credentials (username, password, and domain)*—own credentials settable using the `set_effective_own_gssup_credentials_info()` operation.
- *Asserted identity (username only)*—received credentials settable using operations from `IT_CSI::CSICurrent2`.

The `set_effective_own_gssup_credentials_info()` operation is completely independent of the functionality provided by the `SecurityLevel2::PrincipalAuthenticator` interface and does NOT update the `SecurityLevel2::SecurityManager::own_credentials_list`. This

---

operation might typically be called by an intermediate CORBA routing application that does not want to perform the CSiv2 authentication itself (and subsequently propagate an identity token).

---

**WARNING:** It is critical that the caller understands that the supplied username/password will be transmitted on the wire (directly as supplied).

---

A programmer can access an `IT_CSI::CSICurrent3` object from within an operation context using the following code:

```
// Java
com.iona.corba.IT_CSI.CSICurrent3 it_csi_current = null;
try {
    org.omg.CORBA.Object objref =
        orb.resolve_initial_references("SecurityCurrent");

    it_csi_current =
        com.iona.corba.IT_CSI.CSICurrent3Helper.narrow(objref);
}
catch (org.omg.CORBA.ORBPackage.InvalidName ex) {
    // Error: resolve_initial_references() call failed...
}
catch (org.omg.CORBA.BAD_PARAM ex) {
    // Error: narrow() call failed...
}
```

### **CSICurrent3::set\_effective\_own\_gssup\_credentials\_info()**

```
CredentialsListObject
set_effective_own_gssup_credentials_info(
    in string    username,
    in string    password,
    in string    domain
);
```

---

Sets the GSSUP username, password and domain attributes in the effective own credentials in the current execution context. The supplied username, password and domain are then transmitted on the wire—using the *CS/authentication over transport* mechanism—with every operation invoked inside the current execution context.

This operation is only of relevance in the context of a CSiv2 invocation and effectively overrides the settings for the `SecurityLevel2::InvocationCredentialsPolicy` and the `SecurityLevel2::SecurityManager::own_credentials_list`.

### Parameters

This operation takes the following parameters:

<code>username</code>	The GSSUP username to be transmitted.
<code>password</code>	The GSSUP password to be transmitted.
<code>domain</code>	The GSSUP domain to be transmitted.

### **CSICurrent3::set\_effective\_own\_credentials\_list()**

```
void
set_effective_own_credentials_list(
    in CredentialsListObject    effective_own_credentials
);
```

Reserved for future use.





# IT\_CSI::CSIReceivedCredentials Interface

```
local interface CSIReceivedCredentials :
    IT_TLS_API::TLSReceivedCredentials, CSICredentials
{
    readonly attribute CSICredentials gssup_credentials;
    readonly attribute CSICredentials
        propagated_identity_credentials;
    readonly attribute SecurityLevel2::Credentials
        transport_credentials;
};
```

The `CSIReceivedCredentials` interface, which inherits from `IT_TLS_API::TLSReceivedCredentials` and `SecurityLevel2::ReceivedCredentials`. The `OMG SecurityLevel2::Current::received_credentials()` operation returns a single `SecurityLevel2::ReceivedCredentials` object. However a CSIV2 server may receive as many as three credentials from a CSI client:

- Transport TLS credentials
- Propagated identity credentials
- Authenticated credentials over the transport.

The `CSIReceivedCredentials` interface provides access to all three credentials.

The `SecurityLevel2::Current::received_credentials()` operation returns the following credentials type

- Propagated identity credentials, if present
- Authenticated credentials over the transport, if present and propagated identity credentials are not.
- Transport TLS credentials, if present and the above two are not.

## Examples

The following Java example shows how to access the GSSUP credentials received through the CSIV2 authentication over transport mechanism:

```
// Java
```

---

```

import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.Security.AttributeType;
import org.omg.Security.SecAttribute;
import org.omg.SecurityLevel2.Current;
import org.omg.SecurityLevel2.ReceivedCredentials;

import org.omg.Security.*;
import org.omg.SecurityLevel2.*;

import com.iona.corba.tls.cert.*;
import com.iona.corba.IT_CSI.CSIReceivedCredentials;
import com.iona.corba.IT_CSI.CSIReceivedCredentialsHelper;
import com.iona.corba.IT_CSI.CSICredentialsType;
...
//-----
// Access GSSUP Received Credentials -
// this code can be used in the invocation context of a
// secure operation, to access the GSSUP identity received
// via the CSIV2 'authentication over transport' mechanism.
//
org.omg.SecurityLevel2.Current current = null;

try {
    current = (org.omg.SecurityLevel2.Current)
        m_orb.resolve_initial_references("SecurityCurrent");
}
catch(org.omg.CORBA.ORBPackage.InvalidName e) {
    ... // Error: SecurityCurrent initial reference not available.
}

ReceivedCredentials rec_creds = current.received_credentials();
if(rec_creds==null) {
    ... // Error: Received credentials are null."
}
CSISecAttribute csi_rec_creds
    = CSISecAttributeHelper.narrow(rec_creds);
Credentials gssup_credentials_rec
    = csi_rec_creds.gssup_credentials();
SecAttribute[] gssup_attribute
    = gssup_credentials_rec.get_attributes(attributes_types);

if ((gssup_attribute==null) || (gssup_attribute.length==0)) {

```

---

```
    ... // Error: Operation called by user with no GSSUP creds
}

// The GSSUP access ID string is just the authenticated username.
String gssup_access_id = new String(
    gssup_attribute[0].value, 0, gssup_attribute[0].value.length
);
```

### **CSIReceivedCredentials::gssup\_credentials Attribute**

```
readonly attribute CSICredentials gssup_credentials;
```

A reference to the GSSUP credentials received using the CSIV2 *authentication over transport* mechanism; or a nil object reference if no credentials of this type were received. To access the credentials' attributes, use the inherited `SecurityLevel2::Credentials::get_attributes()` operation.

#### **See Also**

```
Security::SecAttribute
IT_CSI::CSICredentialsType
```

### **CSIReceivedCredentials::propagated\_identity\_credentials Attribute**

```
readonly attribute CSICredentials propagated_identity_credentials;
```

A reference to the GSSUP credentials received using the CSIV2 *identity assertion* (principal propagation) mechanism; or a nil object reference if no credentials of this type were received. To access the credentials' attributes, use the inherited `SecurityLevel2::Credentials::get_attributes()` operation.

#### **See Also**

```
Security::SecAttribute
IT_CSI::CSICredentialsType
```

### **CSIReceivedCredentials::transport\_credentials Attribute**

```
readonly attribute SecurityLevel2::Credentials
    transport_credentials;
```

---

A reference to the credentials received through the SSL/TLS transport layer; or a nil object reference if no credentials of this type were received. These credentials normally take the form of an X.509 certificate chain. To access the credentials' attributes, use the `SecurityLevel2::Credentials::get_attributes()` operation.

**See Also**

`Security::SecAttribute`  
`IT_CSI::CSICredentialsType`  
`IT_Certificate::X509CertChain`

# IT\_EventChannelAdmin Module

Module `IT_EventChannelAdmin` describes extensions to the module `CosEventChannelAdmin`. It defines an interface, `EventChannelFactory`, for creating or discovering `EventChannel` objects.

## IT\_EventChannelAdmin Data Types

### IT\_EventChannelAdmin::ChannelID Type

```
typedef long ChannelID;
```

The `ChannelID` is used by the event service to track event channels. This number is assigned by the service when a new event channel is created.

### IT\_EventChannelAdmin::EventChannelInfo Structure

```
struct EventChannelInfo
{
    string                name;
    ChannelID            id;
    CosEventChannelAdmin::EventChannel reference;
};
```

The `EventChannelInfo` is the unit of information managed by the `EventChannelFactory` for a given `EventChannel` instance. `name` is used for administrative purposes.

### IT\_EventChannelAdmin::EventChannelInfoList Sequence

```
typedef sequence<EventChannelInfo> EventChannelInfoList;
```

The `EventChannelInfoList` contains a sequence of `EventChannelInfo` and is the unit returned by `EventChannelFactory::list_channels()`.

---

## IT\_EventChannelAdmin Exceptions

### **IT\_EventChannelAdmin::ChannelAlreadyExists**

```
exception ChannelAlreadyExists {string name;};
```

`ChannelAlreadyExists` is raised when an attempt is made to create an event channel with a name that is already in use. It returns with the name of the channel.

### **IT\_EventChannelAdmin::ChannelNotFound**

```
exception ChannelNotFound {string name;};
```

`ChannelNotFound` is raised when a call to either `EventChannelFactory::find_channel()` or `EventChannelFactory::find_channel_by_id()` cannot find the specified channel. It returns with the name of the specified channel.

# IT\_EventChannelAdmin::EventChannelFactory Interface

```
interface EventChannelFactory : IT_MessagingAdmin::Manager
{
    CosEventChannelAdmin::EventChannel create_channel(
                                        in string      name,
                                        out ChannelID  id)

    raises (ChannelAlreadyExists);

    CosEventChannelAdmin::EventChannel find_channel(
                                        in string      name,
                                        out ChannelID  id)

    raises (ChannelNotFound);

    CosEventChannelAdmin::EventChannel find_channel_by_id(
                                        in ChannelID  id,
                                        out string     name)

    raises (ChannelNotFound);

    EventChannelInfoList list_channels();
};
```

The `EventChannelFactory` interface defines operations for creating and managing untyped event channels. By inheriting from the `Manager` interface, it also has the ability to gracefully shut down the event service.

## EventChannelFactory::create\_channel()

```
//IDL
CosEventChannelAdmin::EventChannel create_channel(
                                        in string      name,
                                        out ChannelID  id)

raises (ChannelAlreadyExists);
```

Creates a new instance of the event service style event channel

---

## Parameters

name	The name of the channel to be created
id	The id of the created channel

### EventChannelFactory::find\_channel()

```
//IDL
CosEventChannelAdmin::EventChannel find_channel(
    in string name,
    out ChannelID id)

raises (ChannelNotFound);
```

Returns an `EventChannel` instance specified by the provided name.

## Parameters

name	The name of the channel
id	The channel id as returned from <code>create_channel()</code>

### EventChannelFactory::find\_channel\_by\_id()

```
//IDL
CosEventChannelAdmin::EventChannel find_channel_by_id(
    in ChannelID id,
    out string name)

raises (ChannelNotFound);
```

Returns an `EventChannel` instance specified by the provided id.

## Parameters

id	The channel id as returned from <code>create_channel()</code>
name	The name of the channel

### EventChannelFactory::list\_channels()

```
//IDL
EventChannelInfoList list_channels();
```



## IT\_EventChannelAdmin::EventChannelFactory Interface

---

Return a list of the `EventChannel` instances associated with the event service.



# IT\_FPS Module

The `IT_FPS` module defines the constants and interface for the `InterdictionPolicy`.

```
const unsigned long FPS_POLICY_BASE =
    IT_PolicyBase::IONA_POLICY_ID + 40;

const CORBA::PolicyType INTERDICTION_POLICY_ID = FPS_POLICY_BASE;

enum InterdictionPolicyValue
{
    DISABLE,
    ENABLE
};

local interface InterdictionPolicy : CORBA::Policy
{
    readonly attribute InterdictionPolicyValue value;
};
```

## **FPS\_POLICY\_BASE Constant**

```
const unsigned long FPS_POLICY_BASE =
    IT_PolicyBase::IONA_POLICY_ID + 40;
```

Specifies the offset used to identify the `InterdictionPolicy`.

## **INTERDICTION\_POLICY\_ID Constant**

```
const CORBA::PolicyType INTERDICTION_POLICY_ID = FPS_POLICY_BASE;
```

Specifies the ID passed to `create_policy()` when creating an `InterdictionPolicy`.

---

## InterdictionPolicyValue Enum

```
enum InterdictionPolicyValue
{
    DISABLE,
    ENABLE
};
```

Specifies the possible values for the `InterdictionPolicy`. The values are defined as follows:

<code>ENABLE</code>	This is the default behavior of the firewall proxy service plug-in. A POA with its <code>InterdictionPolicy</code> set to <code>ENABLE</code> will be proxified.
<code>DISABLE</code>	This setting tells the firewall proxy service plug-in to not proxify the POA. A POA with its <code>InterdictionPolicy</code> set to <code>DISABLE</code> will not use the firewall proxy service and requests made on objects under its control will come directly from the requesting clients.

# IT\_FPS::InterdictionPolicy Interface

This is an interface for a local policy object derived from [CORBA::Policy](#). You create instances of `InterdictionPolicy` by calling [CORBA::ORB::create\\_policy\(\)](#). It is used to specify if a POA is to be proxified by the firewall proxy service.

```
local interface InterdictionPolicy : CORBA::Policy
{
    readonly attribute InterdictionPolicyValue value;
};
```



# The IT\_GIOP Module

**In this chapter**

This chapter contains the following sections:

<a href="#">Module IT_GIOP</a>	<a href="#">page 906</a>
<a href="#">Interface IT_GIOP::ClientVersionConstraintsPolicy</a>	<a href="#">page 907</a>
<a href="#">Interface IT_GIOP::ClientCodeSetConstraintsPolicy</a>	<a href="#">page 908</a>
<a href="#">Interface IT_GIOP::Current</a>	<a href="#">page 909</a>
<a href="#">Interface IT_GIOP::Current2</a>	<a href="#">page 913</a>

---

# Module IT\_GIOP

**Summary** IONA proprietary IDL module that is used to describe the properties of GIOP connections.

---

## IT\_GIOP::CLIENT\_VERSION\_CONSTRAINTS\_POLICY\_ID

**Summary** Identifies the `IT_GIOP::ClientVersionConstraintsPolicy` policy.

**Description** You can pass this policy ID to the `CORBA::ORB::create_policy()` operation to create an `IT_GIOP::ClientVersionConstraintsPolicy` policy instance.

---

## IT\_GIOP::CLIENT\_CODESET\_CONSTRAINTS\_POLICY\_ID

**Summary** Identifies the `IT_GIOP::ClientCodeSetConstraintsPolicy` policy.

**Description** You can pass this policy ID to the `CORBA::ORB::create_policy()` operation to create an `IT_GIOP::ClientCodeSetConstraintsPolicy` policy instance.

---

## IT\_GIOP::VersionSeq

**Summary** A list of GIOP version numbers.

---

## IT\_GIOP::ClientCodeSetConstraintsPolicyValue

**Summary** A collection of narrow and wide character codesets which the client is restricted to use when opening a new connection.

**Description** IONA-internal use only.



---

## Interface `IT_GIOP::ClientVersionConstraintsPolicy`

**Summary** A policy that limits the GIOP versions a client can use when opening a new connection.

**Description** IONA-internal use only.  
Instead of specifying the client's GIOP version by programming, you can set the relevant configuration variable. To specify the GIOP version, use one of the following configuration variables (`iiop` for insecure IIOP and `iiop_tls` for secure IIOP):

```
plugins:iiop:client_version_policy  
plugins:iiop_tls:client_version_policy
```

---

### `IT_GIOP::ClientVersionConstraintsPolicy::allowed_versions`

**Summary** Returns the list of GIOP versions that the client is constrained to use by this policy.

**Description** IONA-internal use only.

---

## Interface IT\_GIOP::ClientCodeSetConstraintsPolicy

**Summary** A policy that limits the character codesets a client can use when opening a new connection.

**Description** IONA-internal use only.

Instead of specifying the client's codesets by programming, you can set the relevant configuration variables. To specify the native codeset (`ncs`) or conversion codeset (`ccs`) for narrow characters (`char`) or wide characters (`wchar`), use the following configuration variables:

```
plugins:codeset:char:ncs
plugins:codeset:char:ccs
plugins:codeset:wchar:ncs
plugins:codeset:wchar:ccs
```

---

### IT\_GIOP::ClientCodeSetConstraintsPolicy::value

**Summary** Returns the character code sets that the client is constrained to use by this policy.

**Description** IONA-internal use only.

---

## Interface IT\_GIOP::Current

<b>Summary</b>	An object that provides access to miscellaneous attributes of a GIOP connection.
<b>Description</b>	<p>On the client side, the <code>IT_GIOP::Current</code> object is used to set attributes that affect all of the outgoing connections opened in the current thread.</p> <p>On the server side, the <code>IT_GIOP::Current</code> object is used to access the attributes of the incoming GIOP connection (the attributes are only accessible in an invocation context).</p> <p>An instance of <code>IT_GIOP::Current</code> can be obtained by passing the string, <code>IT_GIOPCurrent</code>, to <code>CORBA::ORB::resolve_initial_references()</code>.</p>
<b>Java implementation</b>	<p>To obtain a reference to an <code>IT_GIOP::Current</code> object in Java, use the following code:</p>

```
// Java
com.ionacorba.IT_GIOP.Current giop_current = null;
try {
    org.omg.CORBA.Object objref =
        orb.resolve_initial_references("IT_GIOPCurrent");

    giop_current =
        com.ionacorba.IT_GIOP.CurrentHelper.narrow(objref);
}
catch (org.omg.CORBA.ORBPackage.InvalidName ex) {
    // Error: resolve_initial_references() call failed...
}
catch (org.omg.CORBA.BAD_PARAM ex) {
    // Error: narrow() call failed...
}
```

---

### IT\_GIOP::Current::negotiated\_version

<b>Summary</b>	Returns the negotiated GIOP version used by the current connection.
<b>Description</b>	Available on the server side only. This property is negotiated per-connection.

---

## IT\_GIOP::Current::negotiated\_char\_codeset

<b>Summary</b>	Returns the negotiated narrow character codeset ID used by the current connection.
<b>Description</b>	Available on the server side only. This property is negotiated per-connection.

---

## IT\_GIOP::Current::negotiated\_wchar\_codeset

<b>Summary</b>	Returns the negotiated wide character codeset ID used by the current connection.
<b>Description</b>	Available on the server side only. This property is negotiated per-connection. In Orbix, it is possible for this property to be undefined (for example, if an Orbix client is connected and the client has not yet sent any wide characters).

---

## IT\_GIOP::Current::local\_principal

<b>Summary</b>	Sets the CORBA Principal for sending in client requests in an octet sequence format.
<b>Description</b>	<p>The local principal can be set only on the client side (per-thread). It affects only the client invocations made from the current thread, overriding the default value (Orbix uses the operating system user ID for the Principal by default).</p> <p>The local principal setting has no effect unless the client is configured to use CORBA Principals (that is, <code>policies:giop:interop_policy:send_principal</code> must be true).</p>
<b>See also</b>	<a href="#">IT_GIOP::Current::local_principal_as_string</a>

---

## IT\_GIOP::Current::local\_principal\_as\_string

<b>Summary</b>	Sets the CORBA Principal for sending in client requests in a string format.
----------------	---

<b>Description</b>	The <code>local_principal_as_string</code> attribute accesses or modifies the local principal value in a string format. When you set this attribute, it is implicitly converted to an octet sequence format (which is also accessible through the <code>local_principal</code> attribute).
<b>Java implementation</b>	The Principal string is returned in UTF-8 format.
<b>See also</b>	<a href="#">IT_GIOP::Current::local_principal</a>

---

## IT\_GIOP::Current::received\_principal

<b>Summary</b>	Accesses the CORBA Principal received with a client request in an octet sequence format.
<b>Description</b>	The received principal can be accessed only on the server side.
<b>Java implementation</b>	If the client did not include a Principal in the request message, this attribute returns <code>null</code> .
<b>See also</b>	<a href="#">IT_GIOP::Current::received_principal_as_string</a>

---

## IT\_GIOP::Current::received\_principal\_as\_string

<b>Summary</b>	Accesses the CORBA Principal received with a client request in a string format.
<b>Description</b>	The <code>received_principal_as_string</code> attribute accesses the received principal value in a string format. When you access this attribute, it is implicitly converted from an octet sequence format (which is also accessible through the <code>received_principal</code> attribute).
<b>Java implementation</b>	The Principal string is returned in UTF-8 format.
<b>See also</b>	<a href="#">IT_GIOP::Current::received_principal</a>

---

## IT\_GIOP::Current::received\_request\_length

<b>Summary</b>	Returns the length of the current received request.
----------------	---

<b>Description</b>	<p>The request length returned by this attribute is equal to the sum of the all the message fragment lengths (the 12-byte GIOP message header is not considered to be part of the message length). For example, if the request consists of just one message (that is, no fragmentation), the returned length is equal to the message body length.</p> <p>Available on the server side only. You can access this attribute in the servant implementation, assuming there is an invocation context.</p>
<b>Java implementation</b>	Not implemented.

---

## **IT\_GIOP::Current::sent\_reply\_length**

<b>Summary</b>	Returns the length of the current sent reply.
<b>Description</b>	IONA-internal use only. Available on the server side only.
<b>Java implementation</b>	Not implemented.

---

## Interface IT\_GIOP::Current2

<b>Summary</b>	An object that provides access to miscellaneous attributes of a GIOP connection.
<b>Description</b>	<p>On the client side, the <code>IT_GIOP::Current2</code> object is used to set attributes that affect all of the outgoing connections opened in the current thread.</p> <p>On the server side, the <code>IT_GIOP::Current2</code> object is used to access the attributes of the incoming GIOP connection (the attributes are only accessible in an invocation context).</p> <p>An instance of <code>IT_GIOP::Current2</code> can be obtained by passing the string, <code>IT_GIOPCurrent</code>, to <code>CORBA::ORB::resolve_initial_references()</code>.</p> <p>In a future release, the attributes defined in this interface are likely to be either folded into the base interface, or moved to a more general interface.</p>
<b>Java implementation</b>	<p>To obtain a reference to an <code>IT_GIOP::Current2</code> object in Java, use the following code:</p> <pre>// Java com.ionacorba.IT_GIOP.Current2 giop_current2 = null; try {     org.omg.CORBA.Object objref =         orb.resolve_initial_references("IT_GIOPCurrent");     giop_current2 =         com.ionacorba.IT_GIOP.Current2Helper.narrow(objref); } catch (org.omg.CORBA.ORBPackage.InvalidName ex) {     // Error: resolve_initial_references() call failed... } catch (org.omg.CORBA.BAD_PARAM ex) {     // Error: narrow() call failed... }</pre>

---

### IT\_GIOP::Current2::protocol\_name

<b>Summary</b>	Returns the name of the transport protocol underlying GIOP over which the current request was received.
----------------	---

**Description**

Server side only. This readonly attribute can return one of the following string values:

**Table 17:** *Return Values for the Transport Protocol Name*

Protocol	C++ Return Value	Java Return Value
IIOp	IIOp	iiop
IIOp/TLS	IIOp_TLS	iiop
EGMIOP	EGMIOP	egmiop
SHMIOP	SHMIOP	N/A

---

## IT\_GIOP::Current2::local\_address\_literal

**Summary**

Returns the local address, in string format, of the GIOP connection over which a request was received.

**Description**

Server side only. The format of the returned string depends on the specific protocol being used. For IIOp or IIOp/TLS, it consists of the node address, in IPv4 dotted decimal or IPv6 colon-separated hex notation, followed by a dot and then the decimal port number.

For example, an IPv4 address with host, 127.0.0.1, and IP port, 1234, would be returned as the following string:

```
127.0.0.1.1234
```

An IPv6 address with MAC address, FB:00:5B:97:E5:7D, and IP port, 1234, would be returned as the following string:

```
FB:00:5B:97:E5:7D.1234
```

**See also**

[IT\\_GIOP::Current2::remote\\_address\\_literal](#)

---

## IT\_GIOP::Current2::remote\_address\_literal

**Summary**

Returns the remote address, in string format, of the GIOP connection over which a request was received.



<b>Description</b>	<p>Server side only. The format of the returned string depends on the specific protocol being used. For IIOp or IIOp/TLS, it consists of the node address, in IPv4 dotted decimal or IPv6 colon-separated hex notation, followed by a dot and then the decimal port number.</p> <p>For example, an IPv4 address with host, 127.0.0.1, and IP port, 1234, would be returned as the following string:</p> <pre>127.0.0.1.1234</pre> <p>An IPv6 address with MAC address, FB:00:5B:97:E5:7D, and IP port, 1234, would be returned as the following string:</p> <pre>FB:00:5B:97:E5:7D.1234</pre>
<b>See also</b>	<a href="#">IT_GIOP::Current2::local_address_literal</a>

---

## IT\_GIOP::Current2::local\_address

<b>Summary</b>	Returns the local address, in the form of an object, of the GIOP connection over which a request was received.
<b>Description</b>	<p>IONA-internal use only.</p> <p>Server side only. The type of the returned <code>Object</code> depends on the specific protocol implementation being used, as follows:</p> <ul style="list-style-type: none"> <li>• IIOp protocol—object type is <code>IT_ATLI2_IP::IPAddress</code>.</li> <li>• IIOp/TLS protocol—object type is <code>IT_ATLI2_IP::IPAddress</code>.</li> <li>• SHMIOP protocol—object type is <code>IT_ATLI2_SHM::SHMAddress</code>.</li> <li>• EGMIOP protocol—<i>not implemented</i>.</li> </ul>

---

## IT\_GIOP::Current2::remote\_address

<b>Summary</b>	Returns the remote address, in the form of an object, of the GIOP connection over which a request was received.
<b>Description</b>	<p>IONA-internal use only.</p> <p>Server side only. The type of the returned <code>Object</code> depends on the specific protocol implementation being used, as follows:</p> <ul style="list-style-type: none"> <li>• IIOp protocol—object type is <code>IT_ATLI2_IP::IPAddress</code>.</li> <li>• IIOp/TLS protocol—object type is <code>IT_ATLI2_IP::IPAddress</code>.</li> <li>• SHMIOP protocol—object type is <code>IT_ATLI2_SHM::SHMAddress</code>.</li> </ul>

- EGMIOP protocol—*not implemented*.

# IT\_LoadBalancing Overview

The `IT_LoadBalancing` module provides operations that allow you to organize object references in the naming service into object groups. Object groups provide a means of controlling object load balancing by distributing work across a pool of objects.

- The [ObjectGroup](#) interface provides operations to update object group members.
- The [ObjectGroupFactory](#) interface provides operations to create or locate object groups.

The `IT_LoadBalancing` module also uses the following common data types and exceptions.

**Table 18:** *IT\_LoadBalancing Common Data Types and Exceptions*

Common Data Types	Exceptions
<a href="#">MemberId</a>	<a href="#">NoSuchMember</a>
<a href="#">MemberIdList</a>	<a href="#">DuplicateMember</a>
<a href="#">SelectionMethod</a>	<a href="#">DuplicateGroup</a>
<a href="#">Member</a>	<a href="#">NoSuchGroup</a>
<a href="#">GroupId</a>	
<a href="#">GroupList</a>	

## IT\_LoadBalancing::MemberId Data Type

```
//IDL
typedef string MemberId;
```

An identifying string representing an object group member.

When adding a member to an object group, you must specify a string representing the object. The format of the string is left to the developer. Orbix does not interpret them. The only restriction is that member ids must be unique within each object group.

---

## IT\_LoadBalancing::MemberIdList Data Type

```
//IDL
typedef sequence<MemberId> MemberIdList;
```

A list of member ids that belong to an object group.

## IT\_LoadBalancing::SelectionMethod Data Type

```
//IDL
enum SelectionMethod { ROUND_ROBIN_METHOD, RANDOM_METHOD,
    ACTIVE_METHOD };
```

Specifies the algorithm for mapping a name to a member of an object group.

**ROUND\_ROBIN\_METHOD** Sequentially selects objects from the object group to resolve client requests.

**RANDOM\_METHOD** Randomly selects objects from the object group to resolve client requests.

**ACTIVE\_METHOD** Uses load information supplied by the server or the system administrator to select the object with the lightest load from the object group to resolve client requests.

## IT\_LoadBalancing::Member Data Type

```
//IDL
struct Member
{
    Object obj;
    MemberId id;
};
```

Specifies an object group member.

## IT\_LoadBalancing::GroupId Data Type

```
// IDL
typedef string GroupId;
```

---

A string representing an object group.

When creating an object group, you must specify a string representing the object. The format of the string is left to the developer. Orbix does not interpret them. The only restriction is that group ids must be unique among object groups.

## **IT\_LoadBalancing::GroupList Data Type**

```
//IDL
typedef sequence<GroupId> GroupList;
```

A list of object group ids.

## **IT\_LoadBalancing::NoSuchMember Exception**

```
// IDL
exception NoSuchMember{};
```

Raised when the member id passed to an operation does not specify a member in the current object group.

## **IT\_LoadBalancing::DuplicateMember Exception**

```
// IDL
exception DupliccateMember{};
```

Raised by [IT\\_LoadBalancing::ObjectGroup::add\\_member](#) when the member id identifies a member that is already part of the group.

## **IT\_LoadBalancing::DuplicateGroup Exception**

Raised by [IT\\_LoadBalancing::ObjectGroupFactory::create\\_round\\_robin](#), [IT\\_LoadBalancing::ObjectGroupFactory::create\\_random](#), and [IT\\_LoadBalancing::ObjectGroupFactory::create\\_active](#) when the group id identifies a preexisting group.

---

## **IT\_LoadBalancing::NoSuchGroup Exception**

Raised when the specified group id does not match any registered group.

# IT\_LoadBalancing::ObjectGroup Interface

Object groups are controlled by the `ObjectGroup` interface, which defines the operations for manipulating the members of the object group. An `ObjectGroup` is obtained from an [ObjectGroupFactory](#).

The `ObjectGroup` interface has the following attributes:

- [id](#) contains the group's id string specified when the group is created.
- [selection\\_method](#) specifies which algorithm is used to resolve client requests

The `ObjectGroup` interface has the following operations:

- [pick](#) is called by the naming service to map a client request to an active object.
- [add\\_member\(\)](#) adds an object's reference to an object group.
- [remove\\_member\(\)](#) removes an object's reference from the object group.
- [get\\_member\(\)](#) returns the object by its member id.
- [members\(\)](#) returns a list of all members in the object group.
- [update\\_member\\_load\(\)](#) updates the object's load status.
- [get\\_member\\_load\(\)](#) returns an object's load status.
- [set\\_member\\_timeout\(\)](#) specifies the amount of time between load updates for a specific member. After this time the object will be removed from the group's pool of available objects.
- [get\\_member\\_timeout\(\)](#) returns the member's timeout value.
- [destroy\(\)](#) removes the object group from the naming service.

The complete `ObjectGroup` interface is as follows:

```
interface ObjectGroup {
    readonly attribute string id;
    attribute SelectionMethod selection_method;
    Object pick();
    void add_member (in Member mem)
    raises (DuplicateMember);
```

---

```
void remove\_member (in MemberId id)
raises (NoSuchMember);
Object get\_member (in MemberId id)
raises (NoSuchMember);
MemberIdList members();
void update\_member\_load(in MemberIdList ids, in double curr_load)
raises (NoSuchMember);
double get\_member\_load(in MemberId id)
raises (NoSuchMember);
void set\_member\_timeout(in MemberIdList ids, in long timeout_sec)
raises (NoSuchMember);
long get\_member\_timeout(in MemberId id)
raises (NoSuchMember);
void destroy();
};
```

## ObjectGroup::pick()

```
// IDL
Object pick();
```

Returns an object from the group using the selection algorithm specified when the group was created.

### See Also

```
IT\_LoadBalancing::SelectionMethod,
IT\_LoadBalancing::ObjectGroupFactory::create\_round\_robin\(\),
IT\_LoadBalancing::ObjectGroupFactory::create\_random\(\),
IT\_LoadBalancing::ObjectGroupFactory::create\_active\(\)
```

## ObjectGroup::add\_member()

```
// IDL
void add_member( in Member mem )
raises (DuplicateMember);
```

Adds a reference to an object to the object group and makes it available for picking.



---

## Parameters

`mem` Specifies the object to be added to the object group. It is made up of a `CORBA::Object` and a [MemberId](#).

## Exceptions

[IT\\_LoadBalancing](#): A member with the same [MemberId](#) is already associated with the object group.  
[DuplicateMember](#)

## ObjectGroup::remove\_member()

```
// IDL
void remove_member( in MemberId id )
    raises ( NoSuchMember );
```

Removes the specified object's reference from the object group. It does not effect any other references to the object stored in the naming service.

## Parameters

`id` A string that identifies the object within the object group

## Exceptions

[IT\\_LoadBalancing](#): The specified member does not exist in the object group.  
[NoSuchMember](#)

## ObjectGroup::get\_member()

```
// IDL
Object get_member( in MemberId id )
```

Returns the object specified by `id`.

## Parameters

`id` A string that identifies the object within the object group

---

## Exceptions

[IT\\_LoadBalancing](#) The specified member does not exist in the object group.

[ng::  
NoSuchMember](#)  
[r](#)

## ObjectGroup::members()

```
// IDL  
MemberIdList members();
```

Returns a list containing the ids of all members in the object group.

## ObjectGroup::update\_member\_load()

```
// IDL  
void update_member_load(in MemberIdList ids, in double curr_load)  
raises (NoSuchMember);
```

Specifies the load value used in the [ACTIVE\\_METHOD](#) selection algorithm.

## Parameters

ids	A sequence of <a href="#">MemberId</a> values that specify the objects whose load value is being updated.
curr_load	A double that specifies the load on the specified objects. The higher the value, the higher the load. Using the <a href="#">ACTIVE_METHOD</a> members of the group with the lowest load values are picked first.

## Exceptions

[IT\\_LoadBalancing](#) One or more of the specified members do not exist in the object group.

[ng::  
NoSuchMember](#)  
[r](#)

## See Also

[IT\\_LoadBalancing::SelectionMethod](#),  
[IT\\_LoadBalancing::ObjectGroupFactory::create\\_active\(\)](#),  
[IT\\_LoadBalancing::ObjectGroup::set\\_member\\_timeout\(\)](#)

---

## ObjectGroup::get\_member\_load()

```
// IDL
double get_member_load(in MemberId id)
raises (NoSuchMember);
```

Returns the load value for a specified object.

### Parameters

`id` A string that identifies the object within the object group

### Exceptions

[IT\\_LoadBalancing](#) The specified member does not exist in the object group.

[ng::](#)  
[NoSuchMember](#)  
[r](#)

### See Also

[IT\\_LoadBalancing::ObjectGroup::update\\_member\\_load\(\)](#)

## ObjectGroup::set\_member\_timeout()

```
void set_member_timeout(in MemberIdList ids, in long timeout_sec)
raises (NoSuchMember);
```

Specifies the amount of time, in seconds, that a member has between updates of its load value before it is removed from the list of available objects.

### Parameters

`ids` A sequence of [MemberIds](#) that specify the members whose timeout values are being set.

`timeout_sec` A `long` specifying the number of seconds that an object has between load value updates. After this amount of time has expired the object will be taken off the object groups list of available objects.

### Exceptions

[IT\\_LoadBalancing](#) One or more of the specified members do not exist in the object group.

[ng::](#)  
[NoSuchMember](#)  
[r](#)

---

**See Also** [IT\\_LoadBalancing::ObjectGroup::update\\_member\\_load\(\)](#)

## ObjectGroup::get\_member\_timeout()

```
\\ IDL
long get_member_timeout(in MemberId id)
raises (NoSuchMember);
```

Returns the timeout value for the specified object group member.

### Parameters

id                    A string that identifies the object within the object group

### Exceptions

[IT\\_LoadBalancing::](#) One or more of the specified members do not exist in the  
[ng::](#) object group.  
[NoSuchMember](#)  
[r](#)

**See Also** [IT\\_LoadBalancing::ObjectGroup::set\\_member\\_timeout\(\)](#)

## ObjectGroup::destroy()

```
// IDL
void destroy()
```

Removes the object group from the naming service. Before calling `destroy()` on an object group, you must first [unbind](#) it.

### Exceptions

[CosNaming::](#) The object group is not unbound from the naming service.  
[NamingContext::](#)  
[xt::](#)  
[NotEmpty](#)

**See Also** [CosNaming::NamingContext::unbind\(\)](#)

# IT\_LoadBalancing:: ObjectGroupFactory Interface

The `ObjectGroupFactory` interface provides methods for creating and locating object groups in the naming service.

The `ObjectGroupFactory` interface has the following methods to create object groups:

- [`create\_round\_robin\(\)`](#) creates an object group that uses the [`ROUND\_ROBIN\_METHOD`](#) selection algorithm for picking objects.
- [`create\_random\(\)`](#) creates an object group that uses the [`RANDOM\_METHOD`](#) selection algorithm for picking objects.
- [`create\_active\(\)`](#) creates an object group that uses the [`ACTIVE\_METHOD`](#) selection algorithm for picking objects.

The `ObjectGroupFactory` interface has the following methods for locating object groups in the naming service:

- [`find\_group`](#) returns a specific object group.
- [`rr\_groups`](#) returns a list of all object groups using the [`ROUND\_ROBIN\_METHOD`](#) selection algorithm.
- [`random\_groups`](#) returns a list of all object groups using the [`RANDOM\_METHOD`](#) selection algorithm.
- [`active\_groups`](#) returns a list of all object groups using the [`ACTIVE\_METHOD`](#) selection algorithm.

The complete `ObjectGroupFactory` interface is as follows:

```
interface ObjectGroupFactory {  
    ObjectGroup create_round_robin (in GroupId id)  
        raises (DuplicateGroup);  
    ObjectGroup create_random (in GroupId id)  
        raises (DuplicateGroup);  
    ObjectGroup create_active (in GroupId id)  
        raises (DuplicateGroup);  
    ObjectGroup find_group (in GroupId id)  
        raises (NoSuchGroup);  
}
```

---

```
GroupList rr_groups();  
GroupList random_groups();  
GroupList active_groups();  
};
```

## ObjectGroupFactory::create\_round\_robin()

```
// IDL  
ObjectGroup create_round_robin (in GroupId id)  
  raises (DuplicateGroup);
```

Creates an object group in the naming service. The new group uses the [ROUND\\_ROBIN\\_METHOD](#) selection algorithm for picking objects.

### Parameters

id                    A string identifying the object group. The string must be unique among object groups.

### Exceptions

[IT\\_LoadBalancing](#) The id specified is already in use by another object group.  
[ng::DuplicateGroup](#)

### See Also

[IT\\_LoadBalancing::ROUND\\_ROBIN\\_METHOD](#)

## ObjectGroupFactory::create\_random()

```
ObjectGroup create_random (in GroupId id)  
  raises (DuplicateGroup);
```

Creates an object group in the naming service. The new group uses the [RANDOM\\_METHOD](#) selection algorithm for picking objects.

### Parameters

id                    A string identifying the object group. The string must be unique among object groups.

---

## Exceptions

[IT\\_LoadBalancing::DuplicateGroup](#) The `id` specified is already in use by another object group.

## See Also

[IT\\_LoadBalancing::RANDOM\\_METHOD](#)

## ObjectGroupFactory::create\_active()

```
ObjectGroup create_active (in GroupId id)
  raises (DuplicateGroup);
```

Creates an object group in the naming service. The new group uses the [ACTIVE\\_METHOD](#) selection algorithm for picking objects.

## Parameters

`id` A string identifying the object group. The string must be unique among object groups.

## Exceptions

[IT\\_LoadBalancing::DuplicateGroup](#) The `id` specified is already in use by another object group.

## See Also

[IT\\_LoadBalancing::ACTIVE\\_METHOD](#)

## ObjectGroupFactory::find\_group()

```
//IDL
ObjectGroup find_group (in GroupId id)
  raises (NoSuchGroup);
```

Returns the specified object group.

## Parameters

`id` A string identifying the object group. The string must be unique among object groups.

---

## Exceptions

[IT\\_LoadBalanci](#) The group specified does not exist.

[ng](#) ::

[NoSuchGroup](#)

### **ObjectGroupFactory::rr\_groups()**

```
// IDL
GroupList rr_groups();
```

Returns a sequence of [GroupId](#) that identify all objects groups in the naming service that use [ROUND\\_ROBIN\\_METHOD](#).

### **ObjectGroupFactory::random\_groups()**

```
// IDL
GroupList random_groups();
```

Returns a sequence of [GroupId](#) that identify all objects groups in the naming service that use [RANDOM\\_METHOD](#).

### **ObjectGroupFactory::active\_groups()**

```
// IDL
GroupList random_groups();
```

Returns a sequence of [GroupId](#) that identify all objects groups in the naming service that use [ACTIVE\\_METHOD](#).



# IT\_Logging Overview

The `IT_Logging` module is the centralized point for controlling all logging methods.

- The [EventLog](#) interface controls the reporting of log events.
- The [LogStream](#) interface controls how and where events are received.

The `IT_Logging` module also uses the following common data types, static method, and macros.

**Table 19:** *IT\_Logging Common Data Types, Methods, and Macros*

Common Data Types	Methods and Macros
<a href="#">ApplicationId</a>	<a href="#">format_message()</a>
<a href="#">EventId</a>	
<a href="#">EventParameters</a>	<a href="#">IT_LOG_MESSAGE()</a>
<a href="#">EventPriority</a>	<a href="#">IT_LOG_MESSAGE_1()</a>
<a href="#">SubsystemId</a>	<a href="#">IT_LOG_MESSAGE_2()</a>
<a href="#">Timestamp</a>	<a href="#">IT_LOG_MESSAGE_3()</a>
	<a href="#">IT_LOG_MESSAGE_4()</a>
	<a href="#">IT_LOG_MESSAGE_5()</a>

## IT\_Logging::ApplicationId Data Type

```
//IDL
typedef string ApplicationId;
```

An identifying string representing the application that logged the event.

For example, a Unix and Windows `ApplicationId` contains the host name and process ID (PID) of the reporting process. Because this value can differ from platform to platform, streams should only use it as informational text, and should not attempt to interpret it.

**Enhancement** Orbix enhancement to CORBA.

---

## IT\_Logging::EventId Data Type

```
//IDL
typedef unsigned long EventId;
```

An identifier for the particular event.

**Enhancement** Orbix enhancement to CORBA.

## IT\_Logging::EventParameters Data Type

```
//IDL
typedef CORBA::AnySeq EventParameters;
```

A sequence of locale-independent parameters encoded as a sequence of `Any` values.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging::format\\_message\(\)](#)

## IT\_Logging::EventPriority Data Type

```
//IDL
typedef unsigned short EventPriority;
```

Specifies the priority of a logged event. These can be divided into the following categories of priority.

Information	A significant non-error event has occurred. Examples include server startup/shutdown, object creation/deletion, and information about administrative actions. Informational messages provide a history of events that can be invaluable in diagnosing problems.
Warning	The subsystem has encountered an anomalous condition, but can ignore it and continue functioning. Examples include encountering an invalid parameter, but ignoring it in favor of a default value.

---

Error	An error has occurred. The subsystem will attempt to recover, but may abandon the task at hand. Examples include finding a resource (such as memory) temporarily unavailable, or being unable to process a particular request due to errors in the request.
Fatal Error	An unrecoverable error has occurred. The subsystem or process will terminate.

The possible values for an `EventPriority` consist of the following:

```
LOG_NO_EVENTS
LOG_ALL_EVENTS
LOG_INFO_LOW
LOG_INFO_MED
LOG_INFO_HIGH
LOG_INFO (LOG_INFO_LOW)
LOG_ALL_INFO
```

```
LOG_WARNING
LOG_ERROR
LOG_FATAL_ERROR
```

A single value is used for [EventLog](#) operations that report events or [LogStream](#) operations that receive events. In filtering operations such as [set\\_filter\(\)](#), these values can be combined as a filter mask to control which events are logged at runtime.

**Enhancement** Orbix enhancement to CORBA.

### **IT\_Logging::format\_message()**

Returns a formatted message based on a format description and a sequence of parameters.

**Parameters** Messages are reported in two pieces for internationalization:

<code>description</code>	A locale-dependent string that describes of how to use the sequence of parameters in <code>params</code> .
<code>params</code>	A sequence of locale-dependent parameters.

---

`format_message()` copies the `description` into an output string, interprets each event parameter, and inserts the event parameters into the output string where appropriate. Event parameters that are primitive and `SystemException` parameters are converted to strings before insertion. For all other types, question marks (?) are inserted.

**Enhancement** Orbix enhancement to CORBA.

### **IT\_Logging::SubsystemId Data Type**

```
//IDL
typedef string SubsystemId;
```

An identifying string representing the subsystem from which the event originated. The constant `_DEFAULT` may be used to enable all subsystems.

**Enhancement** Orbix enhancement to CORBA.

### **IT\_Logging::Timestamp Data Type**

```
//IDL
typedef unsigned long Timestamp;
```

The time of the logged event in seconds since January 1, 1970.

**Enhancement** Orbix enhancement to CORBA.

---

## IT\_Logging::PrecisionTimestamp Data Type

```
//IDL
struct PrecisionTimestamp
{
    unsigned long seconds; // The time of the logged event in seconds
                           // since January 1, 1970.
    unsigned long nanoseconds; // The nanoseconds of the logged
                               // event, to the best precision supported by the O/S.
}
```

A timestamp with an accuracy of up to one nanosecond (limited by the precision of the operating system).

**Enhancement** Orbix enhancement to CORBA.

## IT\_LOG\_MESSAGE() Macro

A macro to use for reporting a log message.

### Parameters

event_log	The log ( <a href="#">EventLog</a> ) where the message is to be reported.
subsystem	The <a href="#">SubsystemId</a> .
id	The <a href="#">EventId</a> .
severity	The <a href="#">EventPriority</a> .
desc	A string description of the event.

**Enhancement** Orbix enhancement to CORBA.

**Examples** Here is a simple example of usage:

```
...
IT_LOG_MESSAGE(
    event_log,
    IT_IIOP_Logging::SUBSYSTEM,
    IT_IIOP_Logging::SOCKET_CREATE_FAILED,
    IT_Logging::LOG_ERROR,
    SOCKET_CREATE_FAILED_MSG
);
```

---

## IT\_LOG\_MESSAGE\_1() Macro

A macro to use for reporting a log message with one event parameter.

### Parameters

event_log	The log ( <a href="#">EventLog</a> ) where the message is to be reported.
subsystem	The <a href="#">SubsystemId</a> .
id	The <a href="#">EventId</a> .
severity	The <a href="#">EventPriority</a> .
desc	A string description of the event.
param0	A single parameter for an <a href="#">EventParameters</a> sequence.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging::IT\\_LOG\\_MESSAGE\(\)](#)

## IT\_LOG\_MESSAGE\_2() Macro

A macro to use for reporting a log message with two event parameters.

### Parameters

event_log	The log ( <a href="#">EventLog</a> ) where the message is to be reported.
subsystem	The <a href="#">SubsystemId</a> .
id	The <a href="#">EventId</a> .
severity	The <a href="#">EventPriority</a> .
desc	A string description of the event.
param0	The first parameter for an <a href="#">EventParameters</a> sequence.
param1	The second parameter for an <a href="#">EventParameters</a> sequence.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging::IT\\_LOG\\_MESSAGE\(\)](#)

## IT\_LOG\_MESSAGE\_3() Macro

A macro to use for reporting a log message with three event parameters.

---

## Parameters

event_log	The log ( <a href="#">EventLog</a> ) where the message is to be reported.
subsystem	The <a href="#">SubsystemId</a> .
id	The <a href="#">EventId</a> .
severity	The <a href="#">EventPriority</a> .
desc	A string description of the event.
param0	The first parameter for an <a href="#">EventParameters</a> sequence.
param1	The second parameter for an <a href="#">EventParameters</a> sequence.
param2	The third parameter for an <a href="#">EventParameters</a> sequence.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging](#): : [IT\\_LOG\\_MESSAGE\(\)](#)

## IT\_LOG\_MESSAGE\_4() Macro

A macro to use for reporting a log message with four event parameters.

## Parameters

event_log	The log ( <a href="#">EventLog</a> ) where the message is to be reported.
subsystem	The <a href="#">SubsystemId</a> .
id	The <a href="#">EventId</a> .
severity	The <a href="#">EventPriority</a> .
desc	A string description of the event.
param0	The first parameter for an <a href="#">EventParameters</a> sequence.
param1	The second parameter for an <a href="#">EventParameters</a> sequence.
param2	The third parameter for an <a href="#">EventParameters</a> sequence.
param3	The forth parameter for an <a href="#">EventParameters</a> sequence.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging](#): : [IT\\_LOG\\_MESSAGE\(\)](#)

---

## IT\_LOG\_MESSAGE\_5() Macro

A macro to use for reporting a log message with five event parameters.

### Parameters

event_log	The log ( <a href="#">EventLog</a> ) where the message is to be reported.
subsystem	The <a href="#">SubsystemId</a> .
id	The <a href="#">EventId</a> .
severity	The <a href="#">EventPriority</a> .
desc	A string description of the event.
param0	The first parameter for an <a href="#">EventParameters</a> sequence.
param1	The second parameter for an <a href="#">EventParameters</a> sequence.
param2	The third parameter for an <a href="#">EventParameters</a> sequence.
param3	The forth parameter for an <a href="#">EventParameters</a> sequence.
param4	The fifth parameter for an <a href="#">EventParameters</a> sequence.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging](#): :[IT\\_LOG\\_MESSAGE\(\)](#)



# IT\_Logging::EventLog Interface

Logging is controlled with the `EventLog` interface, which defines operations to register interfaces for receiving notification of logged events, report logged events, and filter logged events. Each ORB maintains its own `EventLog` instance, which applications obtain by calling [`resolve\_initial\_references\(\)`](#) with the string argument `IT_EventLog`.

The `EventLog` interface has the following operations:

- [`register\_stream\(\)`](#) registers the receivers of log events. [`report\_event\(\)`](#) reports log events and [`report\_message\(\)`](#) reports messages to receivers.
- [`get\_filter\(\)`](#), [`set\_filter\(\)`](#), [`expand\_filter\(\)`](#), and [`clear\_filter\(\)`](#) set filters for which log events are reported.

An `EventLog` has several operations for controlling which events are logged at runtime. A filter has an [`EventPriority`](#) that describes the types of events that are reported. Every subsystem is associated with a filter that controls which events are allowed for that subsystem. A default filter is also associated with the entire `EventLog`.

The complete `EventLog` interface is as follows:

```
// IDL in module IT_Logging
interface EventLog {
    void register\_stream\(\)(
        in LogStream the_stream
    );

    void report\_event\(\)(
        in SubsystemId    subsystem,
        in EventId        event,
        in EventPriority  priority,
        in any             event_data
    );

    void report\_message\(\)(
        in SubsystemId    subsystem,
        in EventId        event,
```

---

```
        in EventPriority  priority,
        in string         description,
        in EventParameters parameters
    );

    EventPriority get\_filter(
        in SubsystemId subsystem
    );

    void set\_filter(
        in SubsystemId  subsystem,
        in EventPriority filter_mask
    );

    void expand\_filter(
        in SubsystemId  subsystem,
        in EventPriority filter_mask
    );

    void clear\_filter(
        in SubsystemId subsystem
    );
    ...
};
```

### **EventLog::clear\_filter()**

```
// IDL
void clear_filter(
    in SubsystemId subsystem
);
```

Removes an explicitly configured subsystem filter, causing the subsystem to revert to using the default filter.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging::EventLog::get\\_filter\(\)](#)

---

## EventLog::expand\_filter()

```
// IDL
void expand_filter(
    in SubsystemId subsystem,
    in EventPriority filter_mask
);
```

Adds to a subsystem filter by combining the new filter mask with the existing subsystem filter.

### Parameters

subsystem	The name of the subsystem for which the filter applies.
filter_mask	A value representing the types of events to be reported.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging::EventLog::set\\_filter\(\)](#)  
[IT\\_Logging::EventLog::clear\\_filter\(\)](#)

## EventLog::get\_filter()

```
// IDL
EventPriority get_filter(
    in SubsystemId subsystem
);
```

Returns a sub-system's filter priorities.

### Parameters

subsystem	The name of the subsystem for which the filter applies.
-----------	---

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging::EventLog::get\\_filter\(\)](#)

## EventLog::register\_stream()

```
// IDL
void register_stream(
```

---

```
        in LogStream the_stream
    );
```

Explicitly registers a [LogStream](#).

### Parameters

`the_stream`      The stream to register.

Log events “flow” to receivers on streams, thus streams must be registered with the `EventLog`. Once registered, the stream will receive notification of logged events.

An `EventLog` can have multiple streams registered at one time, and it can have a single stream registered more than once.

**Enhancement**    Orbix enhancement to CORBA.

**See Also**        [IT\\_Logging::LogStream](#)

## EventLog::report\_event()

```
// IDL
void report_event(
    in SubsystemId    subsystem,
    in EventId        event,
    in EventPriority priority,
    in any            event_data
);
```

Reports an event and its event-specific data.

### Parameters

`subsystem`        The name of the subsystem reporting the event.

`event`            The unique ID defining the event.

`priority`        The event priority.

`event_data`      Event-specific data.

**Enhancement**    Orbix enhancement to CORBA.

**See Also**        [IT\\_Logging::EventLog::report\\_message\(\)](#)

---

## EventLog::report\_message()

```
// IDL
void report_message(
    in SubsystemId subsystem,
    in EventId event,
    in EventPriority priority,
    in string description,
    in EventParameters parameters
);
```

Reports an event and message.

### Parameters

subsystem	The name of the subsystem reporting the event.
event	The unique ID defining the event.
priority	The event priority.
description	A string describing the format of parameters.
parameters	A sequence of parameters for the log.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging::EventLog::report\\_event\(\)](#)

## EventLog::set\_filter()

```
// IDL
void set_filter(
    in SubsystemId subsystem,
    in EventPriority filter_mask
);
```

Sets a filter for a given subsystem. This operation overrides the subsystem's existing filter.

---

## Parameters

<code>subsystem</code>	The name of the subsystem for which the filter applies.
<code>filter_mask</code>	A value representing the types of events to be reported.

A subsystem will use the default filter if its filter has not been explicitly configured by a call to `set_filter()`.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging::EventLog::get\\_filter\(\)](#)

# IT\_Logging::LogStream Interface

The `LogStream` interface allows an application to intercept events and write them to some concrete location via a stream. `IT_Logging::EventLog` objects maintain a list of `LogStream` objects. You register a `LogStream` object from an `EventLog` using `register_stream()`. The complete `LogStream` interface is as follows:

```
// IDL in module IT_Logging
interface LogStream {
    void report\_event(
        in ApplicationId    application,
        in SubsystemId      subsystem,
        in EventId          event,
        in EventPriority     priority,
        in Timestamp        event_time,
        in any               event_data
    );

    void report\_message(
        in ApplicationId    application,
        in SubsystemId      subsystem,
        in EventId          event,
        in EventPriority     priority,
        in Timestamp        event_time,
        in string            description,
        in EventParameters  parameters
    );
};
```

These operations are described in detail as follows:

## **LogStream::report\_event()**

```
// IDL
void report_event(
    in ApplicationId    application,
    in SubsystemId      subsystem,
```

---

```

        in EventId          event,
        in EventPriority    priority,
        in Timestamp        event_time,
        in any              event_data
    );

    // Java
    public void report_event(java.lang.String application,
        java.lang.String subsystem,
        int event,
        short priority,
        int event_time,
        Any event_data)

```

Reports an event and its event-specific data to the log stream.

### Parameters

application	An ID representing the reporting application.
subsystem	The name of the subsystem reporting the event.
event	A unique ID defining the event.
priority	The event priority.
event_time	The time when the event occurred.
event_data	Event-specific data.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging::EventLog::report\\_event\(\)](#)  
[IT\\_Logging::LogStream::report\\_message\(\)](#)

### LogStream::report\_message()

```

// IDL
void report_message(
    in ApplicationId    application,
    in SubsystemId      subsystem,
    in EventId          event,
    in EventPriority    priority,
    in Timestamp        event_time,
    in string           description,
    in EventParameters parameters
);

```



---

```
// Java
public void report_message(java.lang.String application,
    java.lang.String subsystem,
    int event,
    short priority,
    int event_time,
    java.lang.String description,
    Any[] parameters)
```

Reports an event and message to the log stream.

### Parameters

<code>application</code>	An ID representing the reporting application.
<code>subsystem</code>	The name of the subsystem reporting the event.
<code>event</code>	The unique ID defining the event.
<code>priority</code>	The event priority.
<code>event_time</code>	The time when the event occurred.
<code>description</code>	A string describing the format of parameters.
<code>parameters</code>	A sequence of parameters for the log.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging::EventLog::report\\_message\(\)](#)  
[IT\\_Logging::LogStream::report\\_event\(\)](#)



# IT\_Logging::PrecisionLogStream Interface

The PrecisionLogStream interface allows an application to intercept events and write them to some concrete location via a stream. [IT\\_Logging::EventLog](#) objects maintain a list of PrecisionLogStream objects. You register a PrecisionLogStream object from an EventLog using [register\\_stream\(\)](#).

PrecisionLogStream, extends LogStream to allow greater time precision.

The complete PrecisionLogStream interface is as follows:

```
// IDL in module IT_Logging
interface PrecisionLogStream {
    void report_event(
        in ApplicationId application,
        in SubsystemId subsystem,
        in EventId event,
        in EventPriority priority,
        in PrecisionTimestamp event_time,
        in any event_data
    );
    void report_message(
        in ApplicationId application,
        in SubsystemId subsystem,
        in EventId event,
        in EventPriority priority,
        in PrecisionTimestamp event_time,
        in string description,
        in EventParameters parameters
    );
};
```

These operations are described in detail as follows:

---

## PrecisionLogStream::report\_event()

```
// IDL
void report_event(
    in ApplicationId    application,
    in SubsystemId     subsystem,
    in EventId          event,
    in EventPriority    priority,
    in PrecisionTimestamp event_time,
    in any              event_data
);

// Java
public void report_event(java.lang.String application,
    java.lang.String subsystem,
    int event,
    short priority,
    int event_time,
    Any event_data)
```

Reports an event and its event-specific data to the log stream.

### Parameters

application	An ID representing the reporting application.
subsystem	The name of the subsystem reporting the event.
event	A unique ID defining the event.
priority	The event priority.
event_time	The time when the event occurred, with nanosecond precision if available.
event_data	Event-specific data.

**Enhancement** Orbix enhancement to CORBA.

**See Also** [IT\\_Logging::EventLog::report\\_event\(\)](#)  
[IT\\_Logging::LogStream::report\\_message\(\)](#)

## PrecisionLogStream::report\_message()

```
// IDL
void report_message(
    in ApplicationId    application,
```

---

```
    in SubsystemId      subsystem,
    in EventId          event,
    in EventPriority    priority,
    in PrecisionTimestamp event_time,
    in string            description,
    in EventParameters parameters
);

// Java
public void report_message(java.lang.String application,
    java.lang.String subsystem,
    int event,
    short priority,
    int event_time,
    java.lang.String description,
    Any[] parameters)
```

Reports an event and message to the log stream.

### Parameters

application	An ID representing the reporting application.
subsystem	The name of the subsystem reporting the event.
event	The unique ID defining the event.
priority	The event priority.
event_time	The time when the event occurred, with nanosecond precision if available.
description	A string describing the format of parameters.
parameters	A sequence of parameters for the log.

**Enhancement** Orbix enhancement to CORBA.

### See Also

[IT\\_Logging::EventLog::report\\_message\(\)](#)  
[IT\\_Logging::LogStream::report\\_event\(\)](#)



# IT\_MessagingAdmin Module

Module `IT_MessagingAdmin` describes the administrative interface for the Event service.





# IT\_MessagingAdmin::Manager Interface

The `Manager` interface provides administrative operations on an event service.

```
//IDL
interface Manager
{
    readonly attribute string name;
    readonly attribute string host;
    void shutdown();
};
```

## **Manager::shutdown()**

```
//IDL
void shutdown();
```

Shuts down an event service.



# The IT\_NamedKey Module

**In this chapter**

---

This chapter contains the following sections:

<a href="#">Module IT_NamedKey</a>	<a href="#">page 958</a>
<a href="#">Interface IT_NamedKey::NamedKeyRegistry</a>	<a href="#">page 959</a>

---

# Module IT\_NamedKey

**Summary** Defines interfaces related to managing named keys (which appear as object identifiers in `corbaloc:` URLs).

**Description** The named key registry is implemented by the Orbix locator service. Servers register key/object reference associations in the named key registry and clients use these keys to retrieve the associated object references. In practice, this module is intended to facilitate defining `corbaloc:` URLs that are human-readable.

**See also** [IT\\_PlainTextKey](#)

---

## IT\_NamedKey::NamedKeyList

**Summary** A list of named key strings.

**Description** This type is used for the return value of the `IT_NamedKey::NamedKeyRegistry::list_text_keys()` operation.

---

## IT\_NamedKey::NAMED\_KEY\_REGISTRY

**Summary** A string used by the locator to identify the named key registry service.

**See also** `IT_Location::Locator::resolve_service()`

---

## Interface `IT_NamedKey::NamedKeyRegistry`

### Summary

Defines operations to register, de-register, and lookup named keys in the named key registry.

### Description

Named keys are used in conjunction with `corbaloc:` URLs to provide a simple way for clients to access CORBA services. A typical `corbaloc:` URL has the following format:

```
corbaloc:iiop:GIOPVersion@Host:Port/Key
```

This format can be explained as follows:

- *GIOPVersion*—the version of GIOP used on the connection. Can be either 1.0, 1.1, or 1.2.
- *Host:Port*—the hostname, *Host*, and IP port, *Port*, of the Orbix locator service (indirect persistence).
- *Key*—a key string previously registered either with the named key registry or with the `plain_text_key` plug-in.

To register an object reference with the named key registry, the server must first obtain an `IT_Location::Locator` instance by passing the string, `IT_Locator`, to `CORBA::ORB::resolve_initial_references()`. Call the operation, `IT_Location::Locator::resolve_service()`, passing the argument, `IT_NamedKey::NAMED_KEY_REGISTRY`, to obtain an `IT_NamedKey::NamedKeyRegistry` instance. The server can then register one or more named keys by calling the `add_text_key()` operation on `IT_NamedKey::NamedKeyRegistry`.

Note: The named key string format used in this interface does *not* support URL escape sequences (the `%` character followed by two hexadecimal digits).

### Java implementation

The following Java code example shows how to obtain a reference to the named key registry and invoke some operations on the registry.

```
// Java
...
// Get the Locator
org.omg.CORBA.Object objref =
    orb.resolve_initial_references("IT_Locator");
com.ionacorba.IT_Location.Locator locator =
    com.ionacorba.IT_Location.LocatorHelper.narrow(objref);

// Get the Named Key registry
```

```

objref = locator.resolve_service(
    com.iona.corba.IT_NamedKey.NAMED_KEY_REGISTRY
);
com.iona.corba.IT_NamedKey.NamedKeyRegistry registry =
    com.iona.corba.IT_NamedKey.NamedKeyRegistryHelper.narrow(
        objref
    );

// Invoke some operations on the registry
try
{
    registry.add_text_key("MyNamedKey", MyCORBAObjectRef);
    objref = registry.find_text_key("MyNamedKey");
    registry.remove_text_key("MyNamedKey");
}
catch
    (com.iona.corba.IT_NamedKey.NamedKeyRegistryPackage.EntryAlrea
    dyExists ex)
{
    // Error: ...
}
catch
    (com.iona.corba.IT_NamedKey.NamedKeyRegistryPackage.EntryNotFo
    und ex)
{
    // Error: ...
}

```

**See also** [IT\\_PlainTextKey::Forwarder](#)

---

## IT\_NamedKey::NamedKeyRegistry::EntryAlreadyExists

<b>Summary</b>	Raised if you attempt to add a named key that clashes with an existing named key in the registry.
<b>Description</b>	The exception's <code>name</code> element contains the string value of the existing named key in the registry.
<b>See also</b>	<a href="#">IT_NamedKey::NamedKeyRegistry::add_text_key()</a>

---

## IT\_NamedKey::NamedKeyRegistry::EntryNotFound

<b>Summary</b>	Raised if a named key could not be found in the registry.
----------------	---

**Description** The exception's `name` element contains the string value of the named key that you were attempting to find.

**See also** [IT\\_NamedKey::NamedKeyRegistry::remove\\_text\\_key\(\)](#)

---

## IT\_NamedKey::NamedKeyRegistry::add\_text\_key()

**Summary** Adds a new entry to the named key registry.

**Description** The specified object reference, `the_object`, is keyed by the named key parameter, `name`. Internally, the named key registry converts the named key string into an octet sequence and stores the value as an octet sequence (as required by the GIOP specification).

**Parameters**

`name`  
A named key in string format (URL escape sequences not supported).

`the_object`  
The object reference associated with the named key.

**Exceptions**

`EntryAlreadyExists`  
Raised if the registry already contains an entry with the given `name`.

---

## IT\_NamedKey::NamedKeyRegistry::remove\_text\_key()

**Summary** Removes a named key from the registry.

**Parameters**

`name`  
A named key in string format (URL escape sequences not supported).

**Exceptions**

`EntryNotFound`  
Raised if the specified key, `name`, does not exist in the registry.

---

## IT\_NamedKey::NamedKeyRegistry::find\_text\_key()

**Summary** Finds the registry entry for a particular named key (in string format).

**Returns** Returns the object reference associated with the specified key.

**Parameters**

`name`

A named key in string format (URL escape sequences not supported).

---

## **IT\_NamedKey::NamedKeyRegistry::find\_octets\_key()**

**Summary**

Finds the registry entry for a particular named key (in octets format).

**Description**

According to the CORBA specification, the native format of a named key is a sequence of octets (binary 8-bit format). This operation enables you look up the registry by specifying the named key in this native format.

**Returns**

Returns the object reference associated with the specified key.

**Parameters**

`octets`

A named key in octets format.

---

## **IT\_NamedKey::NamedKeyRegistry::list\_text\_keys()**

**Summary**

Lists all of the keys currently stored in the named key registry.

**Returns**

A sequence of strings containing all of the named keys currently in the registry.



# IT\_Naming Module

The `IT_Naming` module contains a single interface, [IT\\_NamingContextExt](#), which provides the method used to bind an object group into the naming service.

[IT\\_NamingContextExt](#) extends [CosNaming::NamingContextExt](#) and provides the method [bind\\_object\\_group](#) which binds an object group to an Iona proprietary naming service.



# IT\_Naming::IT\_NamingContextExt Interface

The complete `IT_NamingContextExt` is as follows:

```
// IDL in Module IT_Naming
Interface IT_NamingContextExt : CosNaming::NamingContextExt
{
    readonly attribute IT\_LoadBalancing::ObjectGroupFactory
        og_factory;
    readonly attribute IT\_NamingAdmin::NamingAdmin admin;

    void bind_object_group(
        in CosNaming::Name n,
        in IT\_LoadBalancing::ObjectGroup obj_gp )
    raises ( CosNaming::NamingContext::NotFound,
            CosNaming::NamingContext::CannotProceed,
            CosNaming::NamingContext::InvalidName,
            CosNaming::NamingContext::AlreadyBound );
};
```

## IT\_Naming::IT\_NamingContextExt::bind\_object\_group() Method

Binds an object group to an entry in the naming service.

### Parameters

n	A <a href="#">CosNaming::Name</a> specifying the naming service node to bind the object group to.
obj_gp	The object group to bind into the naming service.

**Enhancement** Orbix enhancement to CORBA.

---

## Exceptions

- [NamingContext::NotFound](#) n did not point to a valid entry in the naming service.
- [NamingContext::CannotProceed](#) The call failed due an internal error.
- [NamingContext::InvalidName](#) n has a sequence length of zero.
- [NamingContext::AlreadyBound](#) obj\_gp is already bound into the naming service

# IT\_NotifyChannelAdmin Module

IONA-proprietary versions of some of the interfaces from  
CosNotifyChannelAdmin.



# IT\_NotifyChannelAdmin::GroupProxyPushSupplier Interface

```
interface GroupProxyPushSupplier :
    CosNotifyChannelAdmin::ProxyPushSupplier
{
    void connect\_group\_any\_push\_consumer(
        in IT\_NotifyComm::GroupPushConsumer group_push_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected,
            CosEventChannelAdmin::TypeError);
};
```

The `GroupProxyPushSupplier` interface supports connections to the channel by endpoint groups receiving events from the channel as untyped `Any`s. Note that such endpoint groups are functionally similar to OMG Event Service push-style consumers of untyped events. The `GroupProxyPushSupplier` interface defined here, however, supports event filtering and configuration of QoS properties in addition to taking advantage of the IP/Multicast message transport.

Through inheritance of the [ProxyPushSupplier](#) interface, the `GroupProxyPushSupplier` interface supports administration of QoS properties, administration of a list of associated filter, mapping filters for event priority and lifetime, and a read-only attribute containing a reference to the [ConsumerAdmin](#) that created it. This inheritance implies that a `GroupProxyPushSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the group's ability to accept a QoS request. The `GroupProxyPushSupplier` interface also inherits a pair of operations that suspend and resume the connection between a `GroupProxyPushSupplier` instance and its associated endpoint group. During the time a connection is suspended, the `GroupProxyPushSupplier` accumulates events destined for the endpoint group but does not transmit them until the connection is resumed.

---

The `GroupProxyPushSupplier` interface inherits the [NotifySubscribe](#) interface defined in [CosNotifyComm](#), enabling it to be notified whenever its associated endpoint group changes the list of event types it is interested in receiving.

The `GroupProxyPushSupplier` interface also inherits from the `PushSupplier` interface defined in `CosEventComm`. This interface supports the operation to disconnect the `GroupProxyPushSupplier` from its associated endpoint group.

The `GroupProxyPushSupplier` interface defines the operation to establish the connection over which the consumer's endpoint group receives events from the channel.

### **GroupProxyPushSupplier:: connect\_group\_any\_push\_consumer()**

```
void connect_group_any_push_consumer(  
    in IT\_NotifyComm::GroupPushConsumer group_push_consumer)  
raises(CosEventChannelAdmin::AlreadyConnected,  
        CosEventChannelAdmin::TypeError);
```

Establishes a connection between an endpoint group of consumers expecting events in the form of `Anys`, and an event. Once the connection is established, the `GroupProxyPushSupplier` sends events to the endpoint group by invoking `push()` on the connected consumer.

#### **Parameters**

`group_push_consumer` The reference to an object supporting the [GroupPushConsumer](#) interface defined in [IT\\_NotifyComm](#). This reference is that of a consumer connecting to the channel for the members of an endpoint group.



---

## Exceptions

<code>AlreadyConnected</code>	Raised if the target object of this operation is already connected to a push consumer object.
<code>TypeError</code>	An implementation of the <a href="#">GroupProxyPushSupplier</a> interface may impose additional requirements on the interface supported by the push consumers in a group (for example, it may be designed to invoke some operation other than <code>push</code> in order to transmit events). If the consumers in the group being connected do not meet those requirements, this operation raises the <code>TypeError</code> exception.



# IT\_NotifyChannelAdmin: GroupSequenceProxyPushSupplier Interface

```
interface GroupSequenceProxyPushSupplier :  
    CosNotifyChannelAdmin::SequenceProxyPushSupplier  
{  
    void connect\_group\_sequence\_push\_consumer(  
        in IT\_NotifyComm::GroupSequencePushConsumer  
        group_push_consumer)  
    raises(CosEventChannelAdmin::AlreadyConnected,  
        CosEventChannelAdmin::TypeError);  
};
```

The `GroupSequenceProxyPushSupplier` interface supports connections to the channel by endpoint groups that receive sequences of structured events from the channel.

Through inheritance of [SequenceProxyPushSupplier](#), the `GroupSequenceProxyPushSupplier` interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the [ConsumerAdmin](#) that created it. This inheritance also implies that a `GroupSequenceProxyPushSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the endpoint group's ability to accept a QoS request. The `GroupSequenceProxyPushSupplier` interface also inherits a pair of operations which suspend and resume the connection between a `GroupSequenceProxyPushSupplier` instance and its associated endpoint group. During the time a connection is suspended, the `GroupSequenceProxyPushSupplier` accumulates events destined for the endpoint group but does not transmit them until the connection is resumed.

The `GroupSequenceProxyPushSupplier` interface also inherits from the [SequencePushSupplier](#) interface defined in [CosNotifyComm](#). This interface supports the operation to close the connection from the endpoint group to the

---

GroupSequenceProxyPushSupplier. Since the [SequencePushSupplier](#) interface inherits from [NotifySubscribe](#), a GroupSequenceProxyPushSupplier can be notified whenever the list of event types that its associated endpoint group is interested in receiving changes.

The GroupSequenceProxyPushSupplier interface defines the operation to establish the connection over which the endpoint group receives events from the channel.

### **GroupSequenceProxyPushSupplier:: connect\_group\_sequence\_push\_consumer()**

```
void connect_group_sequence_push_consumer(  
    in IT\_NotifyComm::GroupSequencePushConsumer  
    group_push_consumer)  
raises(CosEventChannelAdmin::AlreadyConnected,  
    CosEventChannelAdmin::TypeError);
```

Establishes a connection between an endpoint group of consumers expecting sequences of structured events and an event channel. Once the connection is established, the GroupSequenceProxyPushSupplier sends events to its endpoint group by invoking `push_structured_events()` on the connected consumer.

#### **Parameters**

`group_push_consumer` A reference to an object supporting the [GroupSequencePushConsumer](#) interface defined in [IT\\_NotifyComm](#). This reference is that of a consumer connecting to the channel for the members of an endpoint group.

#### **Exceptions**

`AlreadyConnected` Raised if the target object of this operation is already connected to a push consumer.

---

`TypeError`

An implementation of the `GroupSequenceProxyPushSupplier` interface may impose additional requirements on the interface supported by an endpoint group (for example, it may be designed to invoke some operation other than `push_structured_events` in order to transmit events). If the members of the endpoint group being connected do not meet those requirements, this operation raises the `TypeError` exception.



# IT\_NotifyChannelAdmin:: GroupStructuredProxyPushSupplier Interface

```
interface GroupStructuredProxyPushSupplier :  
    CosNotifyChannelAdmin::StructuredProxyPushSupplier  
{  
    void connect\_group\_structured\_push\_consumer(  
        in IT\_NotifyComm::GroupStructuredPushConsumer  
        group_push_consumer)  
        raises(CosEventChannelAdmin::AlreadyConnected,  
            CosEventChannelAdmin::TypeError);  
};
```

The `GroupStructuredProxyPushSupplier` interface supports connections to the channel by endpoint groups that receive structured events from the channel.

Through inheritance of [StructuredProxyPushSupplier](#), the `GroupStructuredProxyPushSupplier` interface supports administration of QoS properties, administration of a list of associated filters, mapping filters for event priority and lifetime, and a read-only attribute containing a reference to the [ConsumerAdmin](#) that created it. This inheritance implies that a `GroupStructuredProxyPushSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the group's ability to accept a QoS request. The `GroupStructuredProxyPushSupplier` interface also inherits a pair of operations to suspend and resume the connection between a `GroupStructuredProxyPushSupplier` instance and its associated endpoint group. During the time a connection is suspended, the `GroupStructuredProxyPushSupplier` accumulates events destined for the endpoint group but does not transmit them until the connection is resumed.

The `GroupStructuredProxyPushSupplier` interface also inherits from the [StructuredPushSupplier](#) interface defined in [CosNotifyComm](#). This interface defines the operation to disconnect the `GroupStructuredProxyPushSupplier`

---

from its associated endpoint group. In addition, the `GroupStructuredProxySupplier` interface inherits from [NotifySubscribe](#), enabling it to be notified whenever its associated endpoint group changes the list of event types it is interested in receiving.

The `GroupStructuredProxyPushSupplier` interface defines the operation to establish the connection over which the consumer's endpoint group receives events from the channel.

### **GroupStructuredProxyPushSupplier:: connect\_group\_structured\_push\_consumer()**

```
void connect_group_structured_push_consumer(  
    in IT\_NotifyComm::GroupStructuredPushConsumer  
    group_push_consumer)  
raises(CosEventChannelAdmin::AlreadyConnected,  
    CosEventChannelAdmin::TypeError );
```

Establishes a connection between an endpoint group of consumers expecting structured events and an event channel. Once the connection is established, the `GroupStructuredProxyPushSupplier` sends events to the endpoint group invoking `push_structured_event()` on the connected consumer.

#### **Parameters**

`group_push_consumer` A reference to an object supporting the [GroupStructuredPushConsumer](#) interface defined in [IT\\_NotifyComm](#). This reference is that of a consumer connecting to the channel for the members of an endpoint group.

#### **Exceptions**

`AlreadyConnected` Raised if the target object of this operation is already connected to a push consumer.



---

`TypeError`

An implementation of the `GroupStructuredProxyPushSupplier` interface may impose additional requirements on the interface supported by an endpoint group (for example, it may be designed to invoke some operation other than `push_structured_event` to transmit events). If the members of the endpoint group being connected do not meet those requirements, this operation raises the `TypeError` exception.



# IT\_NotifyComm Module

An module that defines IONA-proprietary versions of some interfaces from `CosNotifyComm`.



# IT\_NotifyComm::GroupNotifyPublish Interface

```
interface GroupNotifyPublish
{
    oneway void offer_change(
        in CosNotification::EventTypeSeq added,
        in CosNotification::EventTypeSeq removed);
};
```

The `GroupNotifyPublish` interface supports an operation allowing a supplier to announce, or publish, the names of the types of events it supplies. It is an abstract interface which is inherited by all group consumer interfaces, and enables suppliers to inform consumers supporting this interface of the types of events they intend to supply.

When implemented by a group consumer, it allows the consumer to modify its subscription list accordingly.

## GroupNotifyPublish::offer\_change()

```
oneway void offer_change(
    in CosNotification::EventTypeSeq added,
    in CosNotification::EventTypeSeq removed);
```

Allows a supplier of notifications to announce, or publish, the names of the types of events it supplies to consumers using IP/Multicast.

---

**Note:** Each event type name consists of two components: the name of the domain in which the event type has meaning, and the name of the actual event type. Either component of a type name may specify a complete domain/event type name, a domain/event type name containing the wildcard '\*' character, or the special event type name "%ALL".

---

---

## Parameters

added	Sequence of event type names specifying the event types the supplier is adding to the list of event types it plans to supply.
removed	Sequence of event type names specifying the event types which the supplier no longer plans to supply.

# IT\_NotifyComm:: GroupPushConsumer Interface

```
interface GroupPushConsumer : GroupNotifyPublish
{
    oneway void push(in any data);
    oneway void disconnect\_push\_consumer();
};
```

The `GroupPushConsumer` interface supports an operation enabling group consumers to receive unstructured events by the push model. It also defines an operation to disconnect the consumer's endpoint group from its associated proxy supplier. In addition, the `GroupPushConsumer` interface inherits `GroupNotifyPublish` which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

---

**Note:** An object supporting the `GroupPushConsumer` interface can receive all events that are supplied to its associated channel. How events supplied to the channel in other forms are internally mapped into an unstructured event for delivery to a `GroupPushConsumer` is summarized in the *CORBA Notification Service Guide*.

---

## **GroupPushConsumer::push()**

```
oneway void push(in any data);
```

Receives unstructured events by the push model. The implementation of `push()` is application specific, and is supplied by application developers.

---

## Parameters

data	A parameter of type <code>CORBA::Any</code> . Upon invocation, this parameter contains an unstructured event being delivered to the group.
------	--

### **GroupPushConsumer::disconnect\_push\_consumer()**

```
oneway void disconnect_push_consumer();
```

Terminates a connection between the target `GroupPushConsumer` and its associated group proxy supplier. The result of this operation is that the target `GroupPushConsumer` releases all resources allocated to support the connection and disposes of the groups object reference. It also disconnects all other members of the target `GroupPushConsumer`'s endpoint group.



# IT\_NotifyComm:: GroupSequencePushConsumer Interface

```
interface GroupSequencePushConsumer : GroupNotifyPublish
{
    oneway void push\_structured\_events(
        in CosNotification::EventBatch notifications);

    oneway void disconnect\_sequence\_push\_consumer();
};
```

The `GroupSequencePushConsumer` interface supports an operation enabling group consumers to receive sequences of structured events using the push model. It also defines an operation to disconnect the consumer's endpoint group from its associated proxy supplier. The `GroupSequencePushConsumer` interface inherits [GroupNotifyPublish](#) which enabling a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

---

**Note:** An object supporting the `GroupSequencePushConsumer` interface can receive all events which were supplied to its associated channel, including events supplied in a form other than a sequence of structured events. How events supplied to the channel in other forms are internally mapped into a sequence of structured events for delivery to a `GroupSequencePushConsumer` is summarized in the *CORBA Notification Service Guide*.

---

## GroupSequencePushConsumer::push\_structured\_events()

```
oneway void push\_structured\_events(
    in CosNotification::EventBatch notifications);
```

---

Receive sequences of structured events by the push model. The implementation of `push_structured_events` is application specific, and is supplied by application developers.

The maximum number of events that are transmitted within a single invocation of this operation, along with the amount of time a supplier of a sequence of structured events accumulates individual events into the sequence before invoking this operation are controlled by QoS property settings as described in the *CORBA Notification Service Guide*.

## Parameters

`notifications` A parameter of type [EventBatch](#) as defined in [CosNotification](#). Upon invocation, this parameter contains a sequence of structured events being delivered to the group.

## **GroupSequencePushConsumer:: disconnect\_sequence\_push\_consumer()**

```
oneway void disconnect_sequence_push_consumer();
```

Terminates a connection between the target `GroupSequencePushConsumer` and its associated group proxy supplier. The result of this operation is that the target `GroupSequencePushConsumer` releases all resources allocated to support the connection and disposes of the groups object reference. This also disconnects all other members of the target `GroupSequencesPushConsumer`'s endpoint group.

# IT\_NotifyComm:: GroupStructuredPushConsumer Interface

```
interface GroupStructuredPushConsumer : GroupNotifyPublish
{
    oneway void push\_structured\_event(
        in CosNotification::StructuredEvent notification);
    oneway void disconnect\_structured\_push\_consumer();
};
```

The `GroupStructuredPushConsumer` interface supports an operation enabling group consumers to receive structured events by the push model. It also defines an operation to disconnect the push consumer's endpoint group from its associated proxy supplier. In addition, the `GroupStructuredPushConsumer` interface inherits [GroupNotifyPublish](#) which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

---

**Note:** An object supporting the `GroupStructuredPushConsumer` interface can receive all events that were supplied to its associated channel, including events supplied in a form other than a structured event. How events supplied to the channel in other forms are internally mapped into a structured event for delivery to a `GroupStructuredPushConsumer` is summarized in the *CORBA Notification Service Guide*.

---

## **GroupStructuredPushConsumer::push\_structured\_event();**

```
oneway void push\_structured\_event(
    in CosNotification::StructuredEvent notification);
```

---

Receives structured events by the push model. The implementation of `push_structured_event()` is application specific, and is supplied by application developers.

## Parameters

`notification`      A parameter of type [StructuredEvent](#) as defined in [CosNotification](#). Upon invocation, this parameter contains a structured event being delivered to the group.

## **GroupStructuredPushConsumer:: disconnect\_structured\_push\_consumer()**

```
oneway void disconnect_structured_push_consumer();
```

Terminates a connection between the target `GroupStructuredPushConsumer` and its associated group proxy supplier. The result of this operation is that the target `GroupStructuredPushConsumer` releases all resources allocated to support the connection and disposes of the groups object reference. This also disconnects all other members of the target `GroupStructuredPushConsumer`'s endpoint group.

# IT\_NotifyLogAdmin Module

This module extends the OMG specified [NotifyLog](#) and [NotifyLogFactory](#) interfaces to support event subscription and publication. Also provides access to a default filter factory.



# IT\_NotifyLogAdmin::NotifyLog Interface

This interface provides IONA specific extensions to [DsNotifyLogAdmin::NotifyLog](#) to support notification style event publication and subscription.

```
interface NotifyLog :DsNotifyLogAdmin::NotifyLog
{
    CosNotification::EventTypeSeq obtain_offered_types();
    CosNotification::EventTypeSeq obtain_subscribed_types();
};
```

## NotifyLog::obtain\_offered\_types()

[CosNotification::EventTypeSeq](#) obtain\_offered\_types();

Allows event consumers to ascertain what events are being advertised by event suppliers.

## NotifyLog::obtain\_subscribed\_types()

[CosNotification::EventTypeSeq](#) obtain\_subscribed\_types();

Allows event suppliers to ascertain which events the event consumers in the channel are interested in receiving.





# IT\_NotifyLogAdmin::NotifyLogFactory Interface

Extends [DsNotifyLogAdmin::NotifyLogFactory](#) to include a link to the notification channel's default filter factory and a link to the telecom logging service's manager.

```
interface NotifyLogFactory :DsNotifyLogAdmin::NotifyLogFactory
{
  readonly attribute CosNotifyFilter::FilterFactory
    default\_filter\_factory;
  readonly attribute IT_LogAdmin::Manager manager;
};
```

## NotifyLogFactory::default\_filter\_factory Attribute

```
readonly attribute CosNotifyFilter::FilterFactory
  default\_filter\_factory;
```

Provides a reference to the notification channel's default filter factory, which is used to create new filter objects for `NotifyLog` objects.

## NotifyLogFactory::manager Attribute

```
readonly attribute IT_LogAdmin::Manager manager;
```

Provides a link to the telecom logging service's manager.



# The IT\_PlainTextKey Module

**In this chapter**

---

This chapter contains the following sections:

<a href="#">Module IT_PlainTextKey</a>	<a href="#">page 998</a>
<a href="#">Interface IT_PlainTextKey::Forwarder</a>	<a href="#">page 999</a>

---

# Module IT\_PlainTextKey

**Summary**

Defines the interface that accesses the `plain_text_key` plug-in.

**Description**

This module is intended to facilitate defining `corbaloc` URLs that are human-readable. The `plain_text_key` plug-in (part of the `it_art` library) stores a transient list of key/object reference associations and makes this list accessible through the `IT_PlainTextKey::Forwarder` interface.

The `plain_text_key` plug-in is intended to be used in conjunction with *direct persistence* (that is, a server that embeds its own address details into an IOR, so that client connections are made directly to the server, bypassing the locator). By registering a key with the `plain_text_key` plug-in, you can alias a GIOP object ID with a human-readable key. The key can then be used to construct a human-readable `corbaloc` URL.

**See also**

[IT\\_NamedKey](#)

---

## Interface `IT_PlainTextKey::Forwarder`

### Summary

Defines an operation to register a key/object reference entry with the `plain_text_key` plug-in.

### Description

Plain text keys (or named keys) are used in conjunction with `corbaloc:` URLs to provide a simple way for clients to access CORBA services. A typical `corbaloc:` URL has the following format:

```
corbaloc:iiop:GIOPVersion@Host:Port/Key
```

This format can be explained as follows:

- *GIOPVersion*—the version of GIOP used on the connection. Can be either 1.0, 1.1, or 1.2.
- *Host:Port*—the hostname, *Host*, and IP port, *Port*, of the CORBA service (direct persistence).
- *Key*—a key string previously registered either with the `plain_text_key` plug-in or with the named key registry.

To register an object reference with the `plain_text_key` plug-in, the server must obtain an `IT_PlainTextKey::Forwarder` instance by passing the string, `IT_PlainTextKeyForwarder`, to `CORBA::ORB::resolve_initial_references()`. The server can then register one or more named keys by calling the `add_plain_text_key()` operation on the `IT_PlainTextKey::Forwarder` instance.

Note: The key string format used in this interface does *not* support URL escape sequences (the `%` character followed by two hexadecimal digits).

Note: The `plain_text_key` plug-in is intended for use with *direct persistence* (that is, a server that embeds its own address details into an IOR, so that client connections are made directly to the server, bypassing the locator).

### Java implementation

The following Java code shows how to obtain a reference to a plain text key forwarder object and add a new entry.

```
// Java
org.omg.CORBA.Object objref = the_orb.resolve_initial_references(
    "IT_PlainTextKeyForwarder"
);
com.ionacorba.IT_PlainTextKey.Forwarder forwarder =
    com.ionacorba.IT_PlainTextKey.ForwarderHelper.narrow(objref);

forwarder.add_plain_text_key(
```

```

        "MyPlainTextKey",
        MyCORBAObjectReference
    );

```

**See also** [IT\\_NamedKey::NamedKeyRegistry](#)

---

## IT\_PlainTextKey::Forwarder::add\_plain\_text\_key()

<b>Summary</b>	Adds a key/object reference association to a list maintained by the <code>plain_text_key</code> plug-in.
<b>Description</b>	<p>The specified object reference, <code>the_object</code>, is keyed by the key parameter, <code>object_name</code>.</p> <p>Internally, the <code>plain_text_key</code> plug-in converts the named key string into an octet sequence and stores the value as an octet sequence (as required by the GIOP specification).</p>
<b>Parameters</b>	<p><code>object_name</code> A key in string format (URL escape sequences not supported).</p> <p><code>the_object</code> The object reference associated with the key.</p>
<b>See also</b>	<a href="#">IT_NamedKey::NamedKeyRegistry::add_text_key()</a>

# IT\_PortableServer Overview

This module contains Orbix policy enhancements to the `PortableServer` module. The `IT_PortableServer` policies are:

- [ObjectDeactivationPolicy](#)
- [PersistenceModePolicy](#)
- [DispatchWorkQueuePolicy](#)

The `IT_PortableServer` module also contains the following common data structures and constants related to the policies:

- [OBJECT\\_DEACTIVATION\\_POLICY\\_ID](#)
- [ObjectDeactivationPolicyValue](#)
- [PERSISTENCE\\_MODE\\_POLICY\\_ID](#)
- [PersistenceModePolicyValue](#)
- [DISPATCH\\_WORKQUEUE\\_POLICY\\_ID](#)

## **IT\_PortableServer::OBJECT\_DEACTIVATION\_POLICY\_ID Constant**

```
// IDL
const CORBA::PolicyType OBJECT_DEACTIVATION_POLICY_ID = 0x49545F00
    + 1;

// Java
public abstract interface OBJECT_DEACTIVATION_POLICY_ID
public static final int value
```

Defines a policy ID for object deactivation.

**Enhancement** This is an Orbix enhancement.

## **IT\_PortableServer::ObjectDeactivationPolicyValue Enumeration**

```
// IDL
```

---

```
enum ObjectDeactivationPolicyValue {
    DISCARD,
    DELIVER,
    HOLD
};
```

```
// C++
```

An object deactivation policy value. Valid values consist of:

```
DISCARD
DELIVER
HOLD
```

**Enhancement** This is an Orbix enhancement.

**See Also** [IT\\_PortableServer::ObjectDeactivationPolicy](#)

## **IT\_PortableServer::PERSISTENCE\_MODE\_POLICY\_ID Constant**

```
// IDL
const CORBA::PolicyType PERSISTENCE_MODE_POLICY_ID = 0x49545F00 +
    3;
```

Defines a policy ID for the mode of object persistence.

**Enhancement** This is an Orbix enhancement.

## **IT\_PortableServer::PersistenceModePolicyValue Enumeration**

```
// IDL
enum PersistenceModePolicyValue {
    DIRECT_PERSISTENCE,
    INDIRECT_PERSISTENCE
};
```

A persistence mode policy value. Valid values consist of:

```
DIRECT_PERSISTENCE
INDIRECT_PERSISTENCE
```

**Enhancement** This is an Orbix enhancement.



---

**See Also**      [IT\\_PortableServer::PersistenceModePolicy](#)

## **IT\_PortableServer::DISPATCH\_WORKQUEUE\_POLICY\_ID Constant**

```
const CORBA::PolicyType DISPATCH_WORKQUEUE_POLICY_ID =  
IT_PolicyBase::IONA_POLICY_ID + 42;
```

```
// Java  
public abstract interface DISPATCH_WORKQUEUE_POLICY_ID  
public static final int value
```

Defines the policy ID for using WorkQueues to process ORB requests.

**Enhancement**    This is an Orbix enhancement.



# IT\_PortableServer:: DispatchWorkQueuePolicy Interface

This is policy used to specify a `workQueue` to process ORB requests. It is derived from [CORBA::Policy](#). You create instances of the policy by calling [CORBA::ORB::create\\_policy\(\)](#).

```
//IDL
local interface DispatchWorkQueuePolicy : CORBA::Policy
{
    readonly attribute IT\_WorkQueue::WorkQueue workqueue;
}
```



# IT\_PortableServer:: ObjectDeactivationPolicy Class

This is an interface for a local policy object derived from [CORBA::Policy](#). You create instances of `ObjectDeactivationPolicy` by calling [CORBA::ORB::create\\_policy\(\)](#).

```
// IDL
interface ObjectDeactivationPolicy : CORBA::Policy {
    readonly attribute ObjectDeactivationPolicyValue value;
};
```

## **ObjectDeactivationPolicy::value()**

```
// Java
public ObjectDeactivationPolicyValue value()
```

Returns the value of this object deactivation policy.

**Enhancement** This is an Orbix enhancement.



# IT\_PortableServer::PersistenceModePolicy Class

This is an interface for a local policy object derived from [CORBA::Policy](#). You create instances of PersistenceModePolicy by calling [CORBA::ORB::create\\_policy\(\)](#).

```
// IDL
interface PersistenceModePolicy : CORBA::Policy {
    readonly attribute PersistenceModePolicyValue value;
};
```

## PersistenceModePolicy::value()

```
// Java
public PersistenceModePolicyValue value()
```

Returns the value of this persistent mode policy.

**Enhancement** This is an Orbix enhancement.





# IT\_TLS Overview

The `IT_TLS` module defines a single IDL interface, as follows:

- `IT_TLS::CertValidator`

The following data types are defined in the scope of `IT_TLS` to describe certificate validation errors:

- `IT_TLS::CertChainErrorCode` enumeration
- `IT_TLS::CertChainErrorInfo` structure.

## `IT_TLS::CACHE_NONE` Constant

```
const SessionCachingMode CACHE_NONE = 0;
```

A flag that specifies no caching.

**See Also** [IT\\_TLS\\_API::SessionCachingPolicy](#)

## `IT_TLS::CACHE_SERVER` Constant

```
const SessionCachingMode CACHE_SERVER = 0x01;
```

A flag that specifies server-side caching only.

**See Also** [IT\\_TLS\\_API::SessionCachingPolicy](#)

## `IT_TLS::CACHE_CLIENT` Constant

```
const SessionCachingMode CACHE_CLIENT = 0x02;
```

A flag that specifies client-side caching only.

**See Also** [IT\\_TLS\\_API::SessionCachingPolicy](#)

---

## IT\_TLS::CACHE\_SERVER\_AND\_CLIENT Constant

const [SessionCachingMode](#) CACHE\_SERVER\_AND\_CLIENT = 0x04;

A flag that specifies both server-side and client-side caching.

### See Also

[IT\\_TLS\\_API::SessionCachingPolicy](#)

## IT\_TLS::CertChainErrorCode Enumeration

```
//IDL
enum CertChainErrorCode
{
    CERTIFICATE_UNKNOWN,
    CERTIFICATE_DECODE_ERROR,
    CERTIFICATE_SIGNED_BY_UNKNOWN_CA,
    UNSUPPORTED_CERTIFICATE,
    CERTIFICATE_EXPIRED,
    CERTIFICATE_NOT_YET_VALID,
    CERTIFICATE_REVOKED,
    BAD_CERTIFICATE,
    CERTIFICATE_SIGNED_BY_NON_CA_CERTIFICATE,
    CERTIFICATE_CHAIN_TOO_LONG,
    CERTIFICATE_FAILED_CONSTRAINTS_VALIDATION,
    CERTIFICATE_FAILED_APPLICATION_VALIDATION,
    CERTIFICATE_SUBJECT_ISSUER_MISMATCH
};
```

An Orbix-specific error code that gives the reason why a certificate failed to validate.

## IT\_TLS::CertChainErrorInfo Structure

```
//IDL
struct CertChainErrorInfo
{
    short          error_depth;
    string         error_message;

    CertChainErrorCode error_reason;

    // If this field is true then the two subsequent field may be
```

---

```

    // examined to get more detail from the underlying toolkit if
    // required. These are non portable values and are only ever
    // likely to be used for diagnostic purposes.
    boolean external_error_set;
    short external_error_depth;
    long external_error;
    string external_error_string;
};

```

This structure is initialized with error information if a certificate chain fails the validation checks made by Orbix SSL/TLS. Two different levels of error information are generated by the Orbix SSL/TLS runtime:

- Error information generated by Orbix SSL/TLS—provided by the `error_depth`, `error_message`, and `error_reason` members.
- Error information generated by an underlying third-party toolkit—provided by the `external_error_depth`, `external_error`, and `external_error_string` members.

The structure contains the following elements:

<code>error_depth</code>	A positive integer that indexes the chain depth of the certificate causing the error. Zero indicates the peer certificate.
<code>error_message</code>	A descriptive error string (possibly from the lower level toolkit).
<code>error_reason</code>	An Orbix-specific error code.
<code>external_error_set</code>	If TRUE, external error details are provided by the underlying toolkit in the member variables following this one.
<code>external_error_depth</code>	The index of the certificate that caused the error, as counted by the underlying toolkit.
<code>external_error</code>	The error code from the underlying toolkit.
<code>external_error_string</code>	A descriptive error string from the underlying toolkit.

---

## IT\_TLS::CipherSuite Type

```
typedef unsigned long CipherSuite;
```

A type that identifies a cipher suite.

### Values

The following constants of IT\_TLS::CipherSuite type are defined in IT\_TLS:

```
TLS_RSA_WITH_NULL_MD5
TLS_RSA_WITH_NULL_SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_WITH_IDEA_CBC_SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_DSS_WITH_DES_CBC_SHA
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_RSA_WITH_DES_CBC_SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_ANON_EXPORT_WITH_RC4_40_MD5
TLS_DH_ANON_WITH_RC4_128_MD5
TLS_DH_ANON_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_ANON_WITH_DES_CBC_SHA
TLS_DH_ANON_WITH_3DES_EDE_CBC_SHA
TLS_FORTEZZA_DMS_WITH_NULL_SHA
TLS_FORTEZZA_DMS_WITH_FORTEZZA_CBC_SHA
```

## IT\_TLS::CipherSuiteList Sequence

```
typedef sequence<CipherSuite> CipherSuiteList;
```

A list of cipher suites.

---

## IT\_TLS::SessionCachingMode Type

```
typedef unsigned short SessionCachingMode;
```

A type that holds a session caching mode flag.

### See Also

[IT\\_TLS\\_API::SessionCachingPolicy](#)



# IT\_TLS::CertValidator Interface

**IDL**

```
// IDL in module IT_TLS
interface CertValidator
{
    boolean
    validate_cert_chain(
        in boolean                chain_is_valid,
        in IT\_Certificate::X509CertChain cert_chain,
        in CertChainErrorInfo          error_info
    );
};
```

**Description** The `CertValidator` interface is a callback interface that can be used to check the validity of a certificate chain. A developer can provide custom validation for secure associations by implementing the `CertValidator` interface, defining the `validate_cert_chain()` operation to do the checking. The developer then creates an instance of the custom `CertValidator` and registers the callback by setting an [IT\\_TLS\\_API::TLS\\_CERT\\_VALIDATOR\\_POLICY](#) policy.

## `CertValidator::validate_cert_chain()`

**IDL**

```
boolean
validate_cert_chain(
    in boolean                chain_is_valid,
    in IT\_Certificate::X509CertChain cert_chain,
    in CertChainErrorInfo          error_info
);
```

**Description** Returns TRUE if the implementation of `validate_cert_chain()` considers the certificate chain to be valid; otherwise returns FALSE.

---

## Parameters

<code>chain_is_valid</code>	TRUE if the certificate chain has passed the validity checks made automatically by the Orbix SSL/TLS toolkit; otherwise FALSE.
<code>cert_chain</code>	The X.509 certificate chain to be checked.
<code>error_info</code>	If the certificate chain has failed the validity checks made by Orbix SSL/TLS, this parameter provides details of the error in the certificate chain.



# IT\_TLS\_API Overview

The `IT_TLS_API` module defines Orbix-specific security policies and an interface, `TLS`, that acts as a factory for certain kinds of security policy. This module contains the following IDL interfaces:

- [CertConstraintsPolicy](#) Interface
- [CertValidatorPolicy](#) Interface
- [MaxChainLengthPolicy](#) Interface
- [SessionCachingPolicy](#) Interface
- [TrustedCAListPolicy](#) Interface
- [TLS](#) Interface
- `TLSCredentials` Interface
- `TLSReceivedCredentials` Interface
- `TLSTargetCredentials` Interface

Associated with each of the security policies, the `IT_TLS_API` module defines the following policy type constants (of `CORBA::PolicyType` type):

```
IT_TLS_API::TLS_CERT_CONSTRAINTS_POLICY
IT_TLS_API::TLS_CERT_VALIDATOR_POLICY
IT_TLS_API::TLS_MAX_CHAIN_LENGTH_POLICY
IT_TLS_API::TLS_SESSION_CACHING_POLICY
IT_TLS_API::TLS_TRUSTED_CA_LIST_POLICY
```

The `IT_TLS_API` module also defines IDL structures that are used to supply authentication information to the

[PrincipalAuthenticator::authenticate\(\)](#) operation, depending on the authentication method used. The following structures are defined:

- `PasswordAuthData`
- `PEMCertChainFileAuthData`
- `PKCS12DERAuthData`
- `PKCS12FileAuthData`
- `X509CertChainAuthData`
- `PKCS11AuthData`

---

Associated with each of the authentication structures, the `IT_TLS_API` module defines the following authentication method constants (of [Security::AuthenticationMethod](#) type):

**Table 20:** *Authentication Method Constants and Authentication Structures*

Authentication Method Constant	Authentication Structure
<code>IT_TLS_AUTH_METH_PASSWORD</code>	<code>PasswordAuthData</code>
<code>IT_TLS_AUTH_METH_CERT_CHAIN_FILE</code>	<code>PEMCertChainFileAuthData</code>
<code>IT_TLS_AUTH_METH_PKCS12_DER</code>	<code>PKCS12DERAuthData</code>
<code>IT_TLS_AUTH_METH_PKCS12_FILE</code>	<code>PKCS12FileAuthData</code>
<code>IT_TLS_AUTH_METH_CERT_CHAIN</code>	<code>X509CertChainAuthData</code>
<code>IT_TLS_AUTH_METH_PKCS11</code>	<code>PKCS11AuthData</code>

## **IT\_TLS\_API::CertConstraints Sequence**

```
typedef sequence<string> CertConstraints;
```

Holds a list of certificate constraints for a certificate constraints policy.

### **See Also**

`IT_TLS_API::`[CertConstraintsPolicy](#)

## **IT\_TLS\_API::PasswordAuthData**

```
struct PasswordAuthData {  
    string password;  
};
```

Supplies only a password as authentication data.

### **Notes**

Reserved for future use.

---

## IT\_TLS\_API::PEMCertChainFileAuthData

```
struct PEMCertChainFileAuthData {  
    string    password;  
    string    filename;  
};
```

Supplies a password and the file name of a privacy-enhanced mail (PEM) encrypted X.509 certificate chain.

### Notes

Reserved for future use.

## IT\_TLS\_API::PKCS12DERAuthData

```
struct PKCS12DERAuthData {  
    string    password;  
    IT\_Certificate::DERData cert_chain;  
};
```

Supplies a password and a certificate chain in DER format.

### Notes

Reserved for future use.

## IT\_TLS\_API::PKCS12FileAuthData

```
struct PKCS12FileAuthData {  
    string password;  
    string filename;  
};
```

Supplies a password and the file name of a PKCS#12 encrypted X.509 certificate chain. The file name should be an absolute path name.

## IT\_TLS\_API::X509CertChainAuthData

```
struct X509CertChainAuthData {  
    IT\_Certificate::DERData    private_key;  
    IT\_Certificate::X509CertChain cert_chain;  
};
```

Supplies an asymmetric private key and an X.509 certificate chain.

---

## IT\_TLS\_API::PKCS11AuthData

```
struct PKCS11AuthData {  
    string                provider;  
    string                slot;  
    string                pin;  
};
```

Supplies the provider name, slot number, and PIN for a smart card that is accessed through a PKCS #11 interface. In this case, the user's private key and certificate chain are stored on the smart card. The PIN is used to gain access to the smart card.

# IT\_TLS\_API::CertConstraintsPolicy Interface

```
// IDL in module IT_TLS_API
local interface CertConstraintsPolicy : CORBA::Policy
{
    readonly attribute CertConstraints cert_constraints;
};
```

This policy defines a list of constraints to be applied to certificates. This policy type is identified by the [IT\\_TLS\\_API::TLS\\_CERT\\_CONSTRAINTS\\_POLICY](#) policy type constant.

## CertConstraintsPolicy::cert\_constraints Attribute

```
readonly attribute CertConstraints cert_constraints;
```

Holds the list of certificate constraints as a sequence of strings, of [IT\\_TLS\\_API::CertConstraints](#) type.



# IT\_TLS\_API::CertValidatorPolicy Interface

```
// IDL in module IT_TLS_API
local interface CertValidatorPolicy : CORBA::Policy
{
    readonly attribute IT\_TLS::CertValidator    cert_validator;
};
```

This policy can be used to register a customized certificate callback object, of [IT\\_TLS::CertValidator](#) type. This policy type is identified by the [IT\\_TLS\\_API::TLS\\_CERT\\_VALIDATOR\\_POLICY](#) policy type constant.

## CertValidatorPolicy::cert\_validator Attribute

```
readonly attribute IT\_TLS::CertValidator    cert_validator;
```

Holds the customized certificate callback object, of [IT\\_TLS::CertValidator](#) type





# IT\_TLS\_API::MaxChainLengthPolicy Interface

```
// IDL in module IT_TLS_API
local interface MaxChainLengthPolicy : CORBA::Policy
{
    readonly attribute unsigned short max_chain_length;
};
```

This is a simple integer-based policy that controls the maximum certificate chain length permitted. The policy is applicable to servers and clients. This policy type is identified by the [IT\\_TLS\\_API::TLS\\_MAX\\_CHAIN\\_LENGTH\\_POLICY](#) policy type constant.

## Notes

Default is 2.

## MaxChainLengthPolicy::max\_chain\_length Attribute

```
readonly attribute unsigned short max_chain_length;
```

Holds the maximum chain length value.



# IT\_TLS\_API::SessionCachingPolicy Interface

```
// IDL in module IT_TLS_API
local interface SessionCachingPolicy : CORBA::Policy{
    readonly attribute unsigned short cache_mode;
};
```

An Orbix-specific policy to specify the caching mode. This policy applies to clients and servers. This policy type is identified by the [IT\\_TLS\\_API::TLS\\_SESSION\\_CACHING\\_POLICY](#) policy type constant.

Session caching is an Orbix-specific feature that enables secure associations (for example, over TCP/IP connections) to be re-established more quickly after being closed.

To enable session caching for a client-server connection, the client must support client-side caching (`CACHE_CLIENT` or `CACHE_SERVER_AND_CLIENT` policy) and the server must support server-side caching (`CACHE_SERVER` or `CACHE_SERVER_AND_CLIENT` policy). The first time a secure association is established between the client and the server, session information is cached at both ends of the association. If the association is subsequently closed and re-established (as can happen when Automatic Connection Management is enabled), the reconnection occurs more rapidly because some of the steps in the security handshake can be skipped.

The caching optimization is effective only if both client and server are running continuously between the closing and the re-establishment of the connection. Session caching data is not stored persistently and is, therefore, not available to restarted applications.

Each TLS listener uses a separate session cache. For example, if you have two POAs with different `InvocationCredentialsPolicy` values, Orbix SSL/TLS creates a TLS listener and session cache for each POA.

A client will not offer a cached session for reuse to a server if the session was initially created with different effective security policies.

---

## SessionCachingPolicy::cache\_mode Attribute

readonly attribute unsigned short cache\_mode;

Holds the client caching mode. The default value is [IT\\_TLS::CACHE\\_NONE](#).

The values for this policy are as follows:

<a href="#">IT_TLS::CACHE_NONE</a>	No caching.
<a href="#">IT_TLS::CACHE_SERVER</a>	Perform server-side caching only.
<a href="#">IT_TLS::CACHE_CLIENT</a>	Perform client-side caching only.
<a href="#">IT_TLS::CACHE_SERVER_AND_CLIENT</a>	Perform both server-side and client-side caching.

# IT\_TLS\_API::TLS Interface

```
// IDL in module IT_TLS_API
local interface TLS {
    SecurityLevel2::MechanismPolicy
    create_mechanism_policy(
        in IT\_TLS::CipherSuiteList ciphersuite_list
    );
};
```

This interface provides helper operations for the TSL module.

## TLS::create\_mechanism\_policy()

```
SecurityLevel2::MechanismPolicy
create_mechanism_policy(
    in IT\_TLS::CipherSuiteList ciphersuite_list
);
```

Creates a [SecurityLevel2::MechanismPolicy](#) object from a list of ciphersuites, ciphersuite\_list.

**See Also**

[IT\\_TLS::CipherSuite](#)



# IT\_TLS\_API::TLSCredentials Interface

```
// IDL
local interface TLSCredentials : SecurityLevel2::Credentials
{
    IT\_Certificate::X509Cert get_x509_cert();

    IT\_Certificate::X509CertChain get_x509_cert_chain();
};
```

This interface is the base interface for the `IT_TLS_API::TLSReceivedCredentials` and the `IT_TLS_API::TLSTargetCredentials` interfaces. The interface defines operations to retrieve an X.509 certificate chain from the credentials.

## TLSCredentials::get\_x509\_cert()

```
// IDL
IT\_Certificate::X509Cert get_x509_cert();
```

Returns a reference to the X.509 peer certificate (first certificate in the chain) contained in the credentials.

## TLSCredentials::get\_x509\_cert\_chain()

```
// IDL
IT\_Certificate::X509CertChain get_x509_cert_chain();
```

Returns a copy of the X.509 certificate chain contained in the credentials.





# IT\_TLS\_API::TLSReceivedCredentials Interface

```
local interface TLSReceivedCredentials :  
    TLSCredentials,  
    SecurityLevel2::ReceivedCredentials  
{  
};
```

The interface of an Orbix-specific received credentials object, which inherits from the standard `SecurityLevel2::ReceivedCredentials` interface.

`TLSReceivedCredentials` provides extra operations (inherited from `IT_TLS_API::TLSCredentials`) to extract the X.509 certificate chain from the credentials.

An instance of a `TLSReceivedCredentials` object can be obtained by narrowing the `SecurityLevel2::ReceivedCredentials` object reference obtained from the `SecurityLevel2::Current::received_credentials` attribute.



# IT\_TLS\_API::TLSTargetCredentials Interface

```
local interface TLSTargetCredentials :  
    TLSCredentials,  
    SecurityLevel2::TargetCredentials  
{  
};
```

The interface of an Orbix-specific target credentials object, which inherits from the standard `SecurityLevel2::TargetCredentials` interface. `TLSTargetCredentials` provides extra operations (inherited from `IT_TLS_API::TLSCredentials`) to extract the X.509 certificate chain from the credentials.

An instance of a `TLSTargetCredentials` object can be obtained by narrowing the `SecurityLevel2::TargetCredentials` object reference returned from the `SecurityLevel2::SecurityManager::get_target_credentials()` operation.



# IT\_TLS\_API::TrustedCAListPolicy Interface

```
local interface TrustedCAListPolicy : CORBA::Policy
{
    readonly attribute IT\_Certificate::X509CertList
        trusted_ca_list;
};
```

This policy specifies a list of trusted CA certificates. The policy is applicable to both servers and clients. This policy type is identified by the [IT\\_TLS\\_API::TLS\\_TRUSTED\\_CA\\_LIST\\_POLICY](#) policy type constant.

## TrustedCAListPolicy::trusted\_ca\_list Attribute

```
readonly attribute IT\_Certificate::X509CertList trusted_ca_list;
```

Holds the list of trusted CA certificates.



# IT\_TypedEventChannelAdmin Module

Module `IT_TypedEventChannelAdmin` describes extensions to the module `CosTypedEventChannelAdmin`. It defines an interface, `TypedEventChannelFactory`, for creating or discovering `TypedEventChannel` objects.

## IT\_TypedEventChannelAdmin Data Types

### **IT\_TypedEventChannelAdmin::TypedEventChannelInfo Structure**

```
struct TypedEventChannelInfo
{
    string                name;
    IT_EventChannelAdmin::ChannelID    id;
    string                interface_id;
    CosTypedEventChannelAdmin::TypedEventChannel    reference;
};
```

The `TypedEventChannelInfo` is the unit of information managed by the `TypedEventChannelFactory` for a given `TypedEventChannel` instance.

### **IT\_TypedEventChannelAdmin::TypedEventChannelInfoList Sequence**

```
typedef sequence<TypedEventChannelInfo> TypedEventChannelInfoList;
```

The `TypedEventChannelInfoList` contains a sequence of `TypedEventChannelInfo` and is the unit returned by `TypedEventChannelFactory::list_typed_channels()`.





# IT\_TypedEventChannelAdmin:: TypedEventChannelFactory Interface

```
interface TypedEventChannelFactory : IT_MessagingAdmin::Manager
{
    CosTypedEventChannelAdmin::TypedEventChannel
    create_typed_channel(in string name,
                        out IT_EventChannelAdmin::ChannelID id)
    raises (IT_EventChannelAdmin::ChannelAlreadyExists);

    CosTypedEventChannelAdmin::TypedEventChannel
    find_typed_channel(in string name,
                      out IT_EventChannelAdmin::ChannelID id)
    raises (IT_EventChannelAdmin::ChannelNotFound);

    CosTypeEventChannelAdmin::TypedEventChannel
    find_typed_channel_by_id(in IT_EventChannelAdmin::ChannelID id,
                            out string name)
    raises (IT_EventChannelAdmin::ChannelNotFound);

    TypedEventChannelInfoList list_typed_channels();
};
```

The `TypedEventChannelFactory` interface defines operations for creating and managing typed event channels. By inheriting from the `IT_MessagingAdmin::Manager` interface, it also has the ability to gracefully shut down the event service.

## **TypedEventChannelFactory::create\_typed\_channel()**

```
//IDL
CosTypedEventChannelAdmin::TypedEventChannel
    create_typed_channel(in string name,
                        out ITEventChannelAdmin::ChannelID id)
    raises (IT_EventChannelAdmin::ChannelAlreadyExists);
```

Creates a new instance of a typed event channel



## IT\_TypedEventChannelAdmin::TypedEventChannelFactory Interface

---

Return a list of the `TypedEventChannel` instances associated with the event service.



# IT\_WorkQueue Module

The `IT_WorkQueue` module defines the interfaces needed to create and manage user defined work queues.



# IT\_WorkQueue:: AutomaticWorkQueue Interface

```
// IDL
interface AutomaticWorkQueue : WorkQueue
{
    readonly attribute unsigned long threads_total;
    readonly attribute unsigned long threads_working;

    attribute long high\_water\_mark;
    attribute long low\_water\_mark;

    void shutdown(in boolean process_remaining_jobs);
};
```

The AutomaticWorkQueue interface specifies the method used to shutdown an automatic work queue. It also specifies the attributes that limit the size of the queue's thread pool and monitor thread usage.

## **threads\_total Attribute**

```
readonly attribute unsigned long threads_total;
```

The total number of threads in the AutomaticWorkQueue which can process work items. This will indicate how many threads the workqueue currently has if it has been configured to dynamically create and destroy threads as the workload changes.

## **threads\_working Attribute**

```
readonly attribute unsigned long threads_working;
```

Indicates the total number of threads that are busy processing work items at that point in time. This value will vary as the workload of the server changes.

---

## high\_water\_mark Attribute

attribute long high\_water\_mark;

Specifies the maximum number of threads an `AutomaticWorkQueue` instance can have in its active thread pool.

## low\_water\_mark Attribute

attribute long low\_water\_mark;

Specifies the minimum number of threads available to an `AutomaticWorkQueue` instance.

## `AutomaticWorkQueue::shutdown()`

void shutdown(in boolean process\_remaining\_jobs);

Deactivates the queue and releases all resources associated with it.

### Parameters

`process_remaining_jobs` `TRUE` specifies that any items in the queue should be processed before shutting down the queue.

`FALSE` specifies that any items in the queue should be flushed.



# IT\_WorkQueue:: AutomaticWorkQueueFactory Interface

```
// IDL
local interface AutomaticWorkQueueFactory
{
    AutomaticWorkQueue create_work_queue(
        in long          max_size,
        in unsigned long initial_thread_count,
        in long          high\_water\_mark,
        in long          low\_water\_mark);

    AutomaticWorkQueue create_work_queue_with_thread_stack_size(
        in long          max_size,
        in unsigned long initial_thread_count,
        in long          high\_water\_mark,
        in long          low\_water\_mark,
        in long          thread_stack_size);
};
```

The `AutomaticWorkQueueFactory` interface specifies two methods for obtaining an [AutomaticWorkQueue](#). The `AutomaticWorkQueueFactory` is obtained by calling `resolve_initial_references("IT_AutomaticWorkQueueFactory")`.

## **AutomaticWorkQueueFactory::create\_work\_queue()**

```
AutomaticWorkQueue create_work_queue(
    in long          max_size,
    in unsigned long initial_thread_count,
    in long          high\_water\_mark,
    in long          low\_water\_mark);
```

Creates an [AutomaticWorkQueue](#).

---

## Parameters

<code>max_size</code>	The maximum number of items the queue can hold.
<code>initial_thread_count</code>	The initial number of threads the queue has available for processing work items.
<a href="#"><u>high_water_mark</u></a>	The maximum number of threads the queue can generate to process work items.
<a href="#"><u>low_water_mark</u></a>	The minimum number of threads the queue can have available to process work items.

### **AutomaticWorkQueueFactory:: create\_work\_queue\_with\_thread\_stack\_size()**

```
AutomaticWorkQueue create_work_queue_with_thread_stack_size(  
    in long          max_size,  
    in unsigned long initial_thread_count,  
    in long          high\_water\_mark,  
    in long          low\_water\_mark,  
    in long          thread_stack_size);
```

Creates an [AutomaticWorkQueue](#) and specify the size of the thread stack.

## Parameters

<code>max_size</code>	The maximum number of items the queue can hold.
<code>initial_thread_count</code>	The initial number of threads the queue has available for processing work items.
<a href="#"><u>high_water_mark</u></a>	The maximum number of threads the queue can generate to process work items.
<a href="#"><u>low_water_mark</u></a>	The minimum number of threads the queue can have available to process work items.
<code>thread_stack_size</code>	The size, in bytes, of the thread stack used by the queue.

# IT\_WorkQueue::ManualWorkQueue Interface

```
// IDL
interface ManualWorkQueue : WorkQueue
{
    boolean dequeue(out WorkItem work, in long timeout);
    boolean do\_work(in long number_of_jobs, in long timeout);
    void shutdown(in boolean process_remaining_jobs);
};
```

The `ManualWorkQueue` interface specifies the methods for managing a manual work queue.

## ManualWorkQueue::dequeue()

```
boolean dequeue(out WorkItem work, in long timeout);
```

Removes a single [WorkItem](#) from the head of the queue. You must explicitly call [execute\(\)](#) on the [WorkItem](#) to process the request using this method.

### Parameters

<code>work</code>	The <a href="#">WorkItem</a> returned by <code>dequeue()</code> . If the call is unsuccessful, <code>work</code> will be <code>NULL</code> .
<code>timeout</code>	The maximum amount of time the call will block before returning <code>NULL</code> .

## ManualWorkQueue::do\_work()

```
boolean do_work(in long number_of_jobs, in long timeout);
```

Removes the specified number of requests from the queue and processes them. If there are less than the specified number of items on the queue, `do_work()` will block for a specified amount of time to wait for items to be queued.

---

## Parameters

`number_of_jobs` The maximum number of items to process.  
`timeout` The maximum amount of time the call will block before returning.

## **ManualWorkQueue::shutdown()**

```
void shutdown(in boolean process_remaining_jobs);
```

Deactivates the queue and releases all resources associated with it.

## Parameters

`process_remaining_jobs` `TRUE` specifies that any items in the queue should be processed before shutting down the queue.  
`FALSE` specifies that any items in the queue should be flushed.

# IT\_WorkQueue:: ManualWorkQueueFactory Interface

```
// IDL
local interface ManualWorkQueueFactory
{
    ManualWorkQueue create\_work\_queue(in long max_size);
};
```

Defines the method used to obtain a [ManualWorkQueue](#). The `ManualWorkQueueFactory` is obtained by calling `resolve_initial_references("IT_ManualWorkQueueFactory")`.

## **ManualWorkQueueFactory::create\_work\_queue()**

[ManualWorkQueue](#) `create_work_queue`(in long max\_size);

Creates a [ManualWorkQueue](#) object.

### **Parameters**

<code>max_size</code>	Specifies the maximum number of work items the queue can hold.
-----------------------	--



# IT\_WorkQueue::WorkItem Interface

```
// IDL
enum WorkItemStatus
{
    STOP_WORKING,
    CONTINUE_WORKING
};

interface WorkItem
{
    WorkItemStatus execute\(\);
    void destroy\(\);
};
```

The `WorkItem` interface defines requests placed on the work queue. For most purposes, you do not need to implement this interface. The ORB will place requests on the queue and execute them under the covers. You can implement this interface if you want to have additional processing done by the work queues thread pool.

## **WorkItem::execute()**

```
WorkItemStatus execute();
```

Processes the request encapsulated in the [WorkItem](#) object. The only times you need to call this method, is when using a [ManualWorkQueue](#) and removing items from the queue using [dequeue\(\)](#). Also, if you have made a custom [WorkItem](#), you will need to implement this method.

## **WorkItem::Destroy**

```
void destroy();
```

Releases the resources for the current [WorkItem](#) object.





# IT\_WorkQueue::WorkQueue Interface

```
// IDL
interface WorkQueue
{
    readonly attribute long max\_size;
    readonly attribute unsigned long count;

    boolean enqueue(in WorkItem work, in long timeout);
    boolean enqueue\_immediate(in WorkItem work);
    boolean is\_full();
    boolean is\_empty();
    boolean activate();
    boolean deactivate();
    void flush();
    boolean owns\_current\_thread();
};
```

The `WorkQueue` interface defines the base functionality for the [ManualWorkQueue](#) interface and the [AutomaticWorkQueue](#) interface.

## **max\_size attribute**

```
readonly attribute long max_size;
```

Specifies the maximum number of [WorkItems](#) a queue can hold before it is full.

## **WorkQueue::enqueue()**

```
boolean enqueue(in WorkItem work, in long timeout);
```

Places work items into the queue for processing. For CORBA requests, the ORB takes care of placing items into the queue. For custom work items that you wish to handle in the queue, you must explicitly place them on the queue.

---

## Parameters

<code>work</code>	The <a href="#">WorkItem</a> to be placed into the queue.
<code>timeout</code>	The time in seconds that the item will be valid on the queue.

### **WorkQueue::enqueue\_immediate()**

```
boolean enqueue_immediate()
```

Returns `TRUE` and places the work item onto the queue for processing if the work queue is not full and the number of threads is below the high water mark. Effectively, this causes the work item to be processed immediately with out waiting for any current thread to complete. Returns `FALSE` if the work item cannot immediately placed on the work queue.

## Parameters

<code>work</code>	The <a href="#">WorkItem</a> to be placed into the queue.
-------------------	---

### **WorkQueue::is\_full()**

```
boolean is_full();
```

Returns `TRUE` if the [WorkQueue](#) has reached [max\\_size](#). Returns `FALSE` otherwise.

### **WorkQueue::is\_empty()**

```
boolean is_empty();
```

Returns `TRUE` if the [WorkQueue](#) is empty. Returns `FALSE` otherwise.

### **WorkQueue::activate()**

```
boolean activate();
```

Puts the queue into a state where it is ready to receive and process work requests.

---

### **WorkQueue::deactivate()**

```
boolean deactivate();
```

Puts the queue into a state where it will no longer process work requests.

### **WorkQueue::owns\_current\_thread()**

```
boolean owns_current_thread();
```

Returns `TRUE` if the thread making the call is managed by the work queue.

### **WorkQueue::flush()**

```
void flush();
```

Removes all of the items from the queue without processing them.



# IT\_WorkQueue::WorkQueuePolicy Interface

```
// IDL
local interface WorkQueuePolicy : CORBA::Policy
{
    readonly attribute WorkQueue work_queue;
};
```

The `WorkPolicy` interface is the object you pass to `create_policy()` when associating you [WorkQueue](#) with a POA.



# The IT\_ZIOP Module

---

**In this chapter**

This chapter contains the following sections:

<a href="#">Module IT_ZIOP</a>	<a href="#">page 1066</a>
<a href="#">Interface IT_ZIOP::Compressor</a>	<a href="#">page 1068</a>
<a href="#">Interface IT_ZIOP::CompressorFactory</a>	<a href="#">page 1070</a>
<a href="#">Interface IT_ZIOP::CompressionManager</a>	<a href="#">page 1072</a>
<a href="#">Interface IT_ZIOP::CompressionComponent</a>	<a href="#">page 1075</a>
<a href="#">Interface IT_ZIOP::CompressionComponentFactory</a>	<a href="#">page 1076</a>
<a href="#">Interface IT_ZIOP::CompressionEnablingPolicy</a>	<a href="#">page 1077</a>
<a href="#">Interface IT_ZIOP::CompressorIdPolicy</a>	<a href="#">page 1078</a>

---

# Module IT\_ZIOP

<b>Summary</b>	Defines interfaces, exceptions, types and values for the IONA ZIOP Compression plug-in.
<b>Description</b>	This plug-in provides optional compression of all types of GIOP messages through a message-level interceptor that is installed between the GIOP interceptor and the transport interceptor (that is, IIOP, IIOP_TLS, etc). This module defines the plug-in interfaces that register compression algorithms, define the ZIOP IOR Component, and define the Policies associated with compression.

---

## IT\_ZIOP::CompressionException

<b>Summary</b>	Thrown when an error occurs during a compress or decompress operation.
<b>Fields</b>	<code>reason</code> Exception details.

---

## IT\_ZIOP::FactoryAlreadyRegistered

<b>Summary</b>	Thrown if a <code>CompressorFactory</code> with the same <code>CompressorId</code> is already registered with the <code>CompressionManager</code> .
----------------	---

---

## IT\_ZIOP::UnknownCompressorId

<b>Summary</b>	Thrown if a <code>CompressorId</code> is not known.
----------------	---

---

## IT\_ZIOP::CompressorId

<b>Summary</b>	Defines the <code>CompressorId</code> type.
<b>Description</b>	The <code>CompressorId</code> is a unique ID that identifies a particular compression algorithm. Three compression algorithms are defined by the standard ZIOP plug-in: <ul style="list-style-type: none"><li>• <code>gzip</code>—for which ID = 1.</li></ul>



- `pkzip`—for which ID = 2.
  - `bzip2`—for which ID = 3.
- 

## IT\_ZIOP::CompressorFactorySeq

**Summary** A list of `CompressorFactory` objects.

---

## IT\_ZIOP::TAG\_IOR\_ZIOP\_COMPONENT

**Summary** The ZIOP IOR component tag.

**Description** Identifies the ZIOP IOR component, which contains a `ComponentId`.

---

## IT\_ZIOP::COMPRESSION\_ENABLING\_POLICY\_ID

**Summary** The policy ID for the `IT_ZIOP::CompressionEnablingPolicy` policy.

**Description** This constant can be passed as the first argument to the `CORBA::ORB::create_policy()` operation to create a `CompressionEnablingPolicy` instance.

---

## IT\_ZIOP::COMPRESSOR\_ID\_POLICY\_ID

**Summary** The policy ID for the `IT_ZIOP::CompressorIdPolicy` policy.

**Description** This constant can be passed as the first argument to the `CORBA::ORB::create_policy()` operation to create a `CompressorIdPolicy` instance.

---

## Interface IT\_ZIOP::Compressor

**Summary**

Implements a compression algorithm.

**Description**

The key operations of the `Compressor` interface are the `compress()` and `decompress()` operations. Implementing these operations is somewhat complicated by the use of segmented buffers (of `IT_Buffer::Buffer` type).

To give you some idea of how to manipulate a segmented buffer, here is an outline of the steps you would perform to iterate over the bytes in a pre-existing buffer:

- Call `IT_Buffer::Buffer::rewind()` to reset the buffer to the first segment.
- Call `IT_Buffer::Buffer::next_segment()` to get a reference to the first segment in the buffer (of `IT_Buffer::Segment` type).
- Iterate over each byte in the segment (bytes within a segment are contiguous). The first byte of the segment is given by `IT_Buffer::Segment::data + IT_Buffer::Segment::offset`. The last byte of the segment is given by `IT_Buffer::Segment::data + IT_Buffer::Segment::offset + IT_Buffer::Segment::length - 1`.
- Move on to the next segment by calling `IT_Buffer::Buffer::next_segment()`.
- When the last segment is reached, `next_segment()` returns a null pointer.

The `Compressor` object simply performs compression/decompression unconditionally. The logic that determines whether or not it is appropriate to compress/decompress a particular message (based on the effective compression policies) is already built-in to the ZIOP plug-in.

---

### IT\_ZIOP::Compressor::compressor\_factory

**Summary**

The `IT_ZIOP::CompressorFactory` associated with this `Compressor`.

---

## IT\_ZIOP::Compressor::compression\_level

**Summary** The implementation- and algorithm-specific compression level associated with this `Compressor`.

---

## IT\_ZIOP::Compressor::compress()

**Summary** Compresses data from the source `Buffer` into the target `Buffer`.

**Parameters**

`source`

An `IT_Buffer::Buffer` object, which contains the data to compress.

`target`

A non-nil `IT_Buffer::Buffer` object, which should be empty.

**Exceptions**

`IT_ZIOP::CompressionException`

Raised if an error occurs during compression.

---

## IT\_ZIOP::Compressor::decompress()

**Summary** Operation that decompresses data from the source `Buffer` into the target `Buffer`.

**Parameters**

`source`

An `IT_Buffer::Buffer` object, which contains the data to decompress.

`target`

A non-nil `IT_Buffer::Buffer` object, which should be empty.

**Exceptions**

`IT_ZIOP::CompressionException`

Raised if an error occurs during decompression.

---

## Interface `IT_ZIOP::CompressorFactory`

**Summary** A factory for `Compressor` instances with a particular implementation- and algorithm-specific compression level.

**Description**

---

### `IT_ZIOP::CompressorFactory::compressor_id`

**Summary** The `CompressorId` associated with this `CompressorFactory`.

**Description** The compressor ID is a unique identifier for a particular compression algorithm.

---

### `IT_ZIOP::CompressorFactory::compressed_bytes`

**Summary** The total number of compressed bytes read and written by `IT_ZIOP::Compressor` instances created by this `CompressorFactory`.

**Description** That is, this value represents the sum of the lengths of all the `target` arguments of `IT_ZIOP::Compressor::compress()` and all of the `source` arguments of `IT_ZIOP::Compressor::decompress()`.

---

### `IT_ZIOP::CompressorFactory::uncompressed_bytes`

**Summary** The total number of uncompressed bytes read and written by `IT_ZIOP::Compressor` instances created by this `CompressorFactory`.

**Description** That is, this value represents the sum of the lengths of all the `source` arguments of `IT_ZIOP::Compressor::compress()` and all of the `target` arguments of `IT_ZIOP::Compressor::decompress()`.

---

### `IT_ZIOP::CompressorFactory::average_compression`

**Summary** The average compression ratio achieved by the `IT_ZIOP::Compressors` instances created by this `CompressorFactory`.

**Description** The compression ratio is defined as the number of compressed bytes divided by the number of uncompressed bytes (usually a value between 0 and 1).

---

## **IT\_ZIOP::CompressorFactory::get\_compressor()**

**Summary** Creates a new `Compressor` instance or else returns a reference to a pre-existing `Compressor` instance with the given compression level.

**Returns** A new or pre-existing `Compressor` instance that has the same compressor ID as the `CompressorFactory` and a compression level specified by the `compression_level` parameter.

**Parameters** `compression_level`  
An arbitrary parameter that affects the compression algorithm. The interpretation of the `compression_level` parameter is specific to each `Compressor`. In some cases, it might be ignored.

---

## **IT\_ZIOP::CompressorFactory::add\_sample()**

**Summary** Add a sample of compressed and uncompressed bytes.

**Description** Called internally to record the volumes of compressed data and uncompressed data passing through the `Compressor`.

**Parameters** `compressed_bytes`  
The length of the compressed data from the most recently compressed/decompressed message.  
`uncompressed_bytes`  
The length of the uncompressed data from the most recently compressed/decompressed message.

---

## Interface IT\_ZIOP::CompressionManager

<b>Summary</b>	Per-ORB interface to register and unregister <code>IT_ZIOP::CompressorFactory</code> objects.
<b>Description</b>	To obtain a reference to the <code>CompressionManager</code> instance, call the <code>CORBA::ORB::resolve_initial_references()</code> operation with the <code>IT_CompressionManager</code> initial reference string as its argument.

---

### IT\_ZIOP::CompressionManager::register\_factory()

<b>Summary</b>	Register a new <code>CompressorFactory</code> instance.
<b>Java implementation</b>	For example, in Java you can register a compressor factory as follows:

```
// Java
package ziop_compression;

import org.omg.CORBA.Any;
import org.omg.CORBA.ORB;
...
import com.iona.corba.IT_ZIOP.CompressionManager;
import com.iona.corba.IT_ZIOP.CompressionManagerHelper;
...
import java.io.*;

// Setup and Configure the CompressionManager
CompressionManager compression_manager;
org.omg.CORBA.Object ref =
    orb.resolve_initial_references("IT_CompressionManager");
compression_manager = CompressionManagerHelper.narrow(ref);
if(compression_manager == null)
{
    Exception ex = new Exception("Unable to retrieve
        IT_CompressionManager reference");
    ex.printStackTrace();
    throw ex;
}
System.out.println("Registering DemoCompressorFactory with
    Compression Manager");
compression_manager.register_factory(new
    DemoCompressorFactory(100));
```

<b>Parameters</b>	<code>compressor_factory</code> The compressor factory to register.
<b>Exceptions</b>	<code>IT_ZIOP::FactoryAlreadyRegistered</code> Raised if a factory with the same compressor ID has already been registered with this <code>CompressionManager</code> .

---

## **IT\_ZIOP::CompressionManager::unregister\_factory()**

<b>Summary</b>	Unregister a <code>IT_ZIOP::CompressorFactory</code> with the given <code>CompressorId</code> from the <code>CompressionManager</code> .
<b>Parameters</b>	<code>compressor_id</code> The compressor ID that identifies the factory to unregister.
<b>Exceptions</b>	<code>IT_ZIOP::UnknownCompressorId</code> Raised if no factory with the specified compressor ID is registered with the <code>CompressionManager</code> .

---

## **IT\_ZIOP::CompressionManager::get\_factory()**

<b>Summary</b>	Retrieve an <code>IT_ZIOP::CompressorFactory</code> with the given <code>CompressorId</code> from the <code>CompressionManager</code> .
<b>Returns</b>	A reference to the <code>CompressorFactory</code> instance with the specified compressor ID.
<b>Parameters</b>	<code>compressor_id</code> The compressor ID that identifies the factory to retrieve.
<b>Exceptions</b>	<code>IT_ZIOP::UnknownCompressorId</code> Raised if no factory with the specified compressor ID is registered with the <code>CompressionManager</code> .

---

## **IT\_ZIOP::CompressionManager::get\_compressor()**

<b>Summary</b>	Creates a new, or returns a pre-existing, <code>IT_ZIOP::Compressor</code> instance.
<b>Returns</b>	A <code>Compressor</code> instance with the specified compressor ID and compression level.

**Parameters**

`compressor_id`

The compressor ID of the `Compressor` instance to retrieve.

`compression_level`

The compressor level of the `Compressor` instance to retrieve.

**Exceptions**

`IT_ZIOP::UnknownCompressorId`

Raised if no factory with the specified compressor ID is registered with the `CompressionManager`.

---

**IT\_ZIOP::CompressionManager::get\_factories()**

**Summary**

Returns a list of all the registered `CompressorFactory` instances.

**Returns**

A sequence of `IT_ZIOP::CompressorFactory` object references.



---

## Interface `IT_ZIOP::CompressionComponent`

### Summary

The ZIOP IOR Component. Has a `CompressorId` attribute that indicates the compression algorithm supported by the server side.

---

### `IT_ZIOP::CompressionComponent::compressor_id`

### Summary

The compressor ID value from the ZIOP IOR component.

---

## Interface `IT_ZIOP::CompressionComponentFactory`

**Summary** The factory for ZIOP IOR components.

---

### `IT_ZIOP::CompressionComponentFactory::get_compression_component()`

**Summary** Creates ZIOP IOR components for inclusion in server-generated IORs.

**Returns** A new (or flyweighted) `IT_ZIOP::CompressionComponent` object.

**Parameters**  
`compressor_id`  
The compressor ID to embed in the ZIOP IOR component.

---

## Interface IT\_ZIOP::CompressionEnablingPolicy

### Summary

Policy to enable compression using the ZIOP plug-in.

### Description

This policy has a single boolean attribute, indicating if compression is enabled or not.

When the compression enabling policy is set on the *server side*, the server embeds a ZIOP component in the IORs it generates. The presence of a ZIOP component in the IOR indicates to clients that the server is capable of receiving compressed messages. You can set server-side policies at any of the following levels:

- ORB.
- POA.

When the compression enabling policy is set on the *client side*, the client checks IORs for the presence of a ZIOP component. If a ZIOP component is present, the client will attempt to send compressed messages to the server. You can set client-side policies at any of the following levels:

- ORB.
- Thread.
- Object (client proxy).

---

### IT\_ZIOP::CompressionEnablingPolicy::compression\_enabled

#### Summary

Indicates whether this policy enables (`true`) or disables (`false`) compression.

---

## Interface IT\_ZIOP::CompressorIdPolicy

<b>Summary</b>	Policy to specify the compressor ID.
<b>Description</b>	<p>The compressor ID indicates which compression algorithm to use (internally, the compressor ID selects a particular implementation of the <code>IT_ZIOP::Compressor</code> interface).</p> <p>The compressor ID policy can <i>only</i> be set on the server side. The server embeds the compressor ID in a ZIOP component in the IORs that it generates. You can set server-side policies at any of the following levels:</p> <ul style="list-style-type: none"><li>• ORB.</li><li>• POA.</li></ul>

---

### IT\_ZIOP::CompressorIdPolicy::compressor\_id

<b>Summary</b>	Returns the value of the compressor ID represented by this policy instance.
----------------	---

# Messaging Overview

CORBA provides synchronous and deferred synchronous modes of invocations. The Messaging module provides the additional asynchronous mode, also known here as *Asynchronous Method Invocation (AMI)*. The Messaging module includes the following base classes, value types, policy classes, common data structures, and constants:

**Table 21:** *The Messaging Module*

<b>Base Classes and Value Types</b>	<b>Common Structures and Constants</b>	<b>QoS Policy Classes</b>
<a href="#">ExceptionHandler</a>	<a href="#">INVOCATION_POLICIES</a>	<a href="#">RebindPolicy</a>
<a href="#">ReplyHandler</a>	<a href="#">RebindMode</a>	<a href="#">RoutingPolicy</a>
	<a href="#">RoutingType</a>	<a href="#">SyncScopePolicy</a>
	<a href="#">RoutingTypeRange</a>	
	<a href="#">SyncScope</a>	
	<a href="#">TAG_POLICIES</a>	

With synchronous invocations, the client program, or thread, blocks when a remote invocation is made and waits until the results arrive. With deferred synchronous invocations, the client thread continues processing, subsequently polling to see if results are available. Within the CORBA module, the deferred synchronous model is only available when using the Dynamic Invocation Interface.

Many applications require some way of managing remote requests within an asynchronous, event-driven environment in which callbacks are invoked to handle events. Sophisticated applications often need to manage several activities simultaneously, making overlapping remote requests to many objects. This can be achieved using a separate thread for each invocation, but the use of threads considerably raises the application's complexity and the probability of programming errors. The use of threads also creates a resource and synchronization problem in addition to the memory management problem inherent in asynchronous communications.

Messaging provides the *callback model* in which the client passed a callback object reference as part of the invocation. When the reply is available, that callback object is invoked with the data of the reply. The callback model uses a [ReplyHandler](#), which is a CORBA object, implemented by the client application. The [ReplyHandler](#) is passed to an asynchronous method invocation. The [ReplyHandler](#) is invoked when the reply to that request is available.

The [Messaging](#) module also provides a QoS property to help obtain asynchronous behavior. The Messaging QoS includes some [CORBA::Policy](#) derived interfaces for client-side policies to control the behavior of requests and replies. Note however that QoS for method invocations applies to both asynchronous and synchronous invocations. See also the discussion "[Quality of Service Framework](#)".

The following constants and types are available for messaging.

### **Messaging::INVOCATION\_POLICIES Constant**

A service context containing a sequence of quality of service policies in effect for the invocation. The quality of service framework abstract model includes this mechanism for transporting [Policy](#) values as part of interoperable object references and within requests.

### **Messaging::RebindMode Type**

This describes the level of transparent rebinding that may occur during the course of an invocation on an object. Values of type `RebindMode` are used in conjunction with a [RebindPolicy](#). All non-negative values are reserved for use in OMG specifications and include the following constants:

<code>TRANSPARENT</code>	Allows the ORB to silently handle object-forwarding and necessary reconnection during the course of making a remote request.
--------------------------	--

---

NO_REBIND	Allows the ORB to silently handle reopening of closed connections while making a remote request, but prevents any transparent object-forwarding that would cause a change in client-visible effective QoS policies. When the <a href="#">RebindPolicy</a> has this mode in effect, only explicit rebinding is allowed by calling <code>CORBA::Object::_validate_connection()</code> .
NO_RECONNECT	Prevents the ORB from silently handling object-forwards or the reopening of closed connections. When the <a href="#">RebindPolicy</a> has this mode in effect, only explicit rebinding is allowed by calling <code>CORBA::Object::_validate_connection()</code> .

Any negative value for a `RebindMode` is considered a vendor extension.

**See Also**

[Messaging::RebindPolicy](#)

## Messaging::RoutingType Type

Describes the type of routing to be used for invocations on an object reference. `RoutingType` values are used in conjunction with a [RoutingPolicy](#). All non-negative values are reserved for use in OMG specifications and include the following constants:

ROUTE_NONE	Synchronous or deferred synchronous delivery is used. No routers will be used to aid in the delivery of the request.
ROUTE_FORWARD	Asynchronous delivery is used. The request is made through the use of a router and not delivered directly to the target by the client ORB.
ROUTE_STORE_AND_FORWARD	Asynchronous TII is used. The request is made through the use of a router that persistently stores the request before attempting delivery.

Any negative value for a `RoutingType` is considered a vendor extension.

**See Also**

[Messaging::RoutingTypeRange](#)

## Messaging::RoutingTypeRange Structure

This structure describes a range of routing types. It is invalid for the minimum [RoutingType](#) to be greater than the maximum [RoutingType](#).

## Messaging::SyncScope Type

Describes the level of synchronization for a request with respect to the target. Values of type `SyncScope` are used in conjunction with a [SyncScopePolicy](#) to control the behavior of one way operations. All non-negative values are reserved for use in OMG specifications. Any negative value of `SyncScope` is considered a vendor extension. Valid values include:

<code>SYNC_NONE</code>	This is equivalent to one allowable interpretation of CORBA 2.2 oneway operations. The ORB returns control to the client (that is, returns from the method invocation) before passing the request message to the transport protocol. The client is guaranteed not to block. You cannot do location-forwarding with this level of synchronization because no reply is returned from the server.
<code>SYNC_WITH_TRANSPORT</code>	<p>This is equivalent to one allowable interpretation of CORBA 2.2 oneway operations. The ORB returns control to the client only after the transport has accepted the request message. This gives no guarantee that the request will be delivered, but in conjunction with knowledge of the transport it may provide the client with enough assurance.</p> <p>For example, for a direct message over TCP, <code>SYNC_WITH_TRANSPORT</code> is not a stronger guarantee than <a href="#">SYNC_NONE</a>. However, for a store and forward transport, this QoS provides a high level of reliability. You cannot do location-forwarding with this level of synchronization because no reply is returned from the server.</p>



---

SYNC_WITH_SERVER	The server-side ORB shall send a reply before invoking the target implementation. If a reply of NO_EXCEPTION is sent, any necessary location-forwarding has already occurred. Upon receipt of this reply, the client-side ORB returns control to the client application. This form of guarantee is useful where the reliability of the network is substantially lower than that of the server. The client blocks until all location-forwarding has been completed. For a server using a POA, the reply would be sent after invoking any <i>ServantManager</i> , but before delivering the request to the target Servant.
SYNC_WITH_TARGET	Equivalent to a synchronous, non-oneway operation in CORBA 2.2. The server-side ORB shall only send the reply message after the target has completed the invoked operation. Note that any LOCATION_FORWARD reply will already have been sent prior to invoking the target and that a SYSTEM_EXCEPTION reply may be sent at anytime (depending on the semantics of the exception). Even though it was declared oneway, the operation actually has the behavior of a synchronous operation. This form of synchronization guarantees that the client knows that the target has seen and acted upon a request. the OTS can only be used with this highest level of synchronization. Any operations invoked with lesser synchronization precludes the target from participating in the client's current transaction.

**See Also**

[Messaging::SyncScopePolicy](#)

## Messaging::TAG\_POLICIES Constant

A profile component containing the sequence of quality of service policies exported with the object reference by an object adapter. The quality of service framework abstract model includes this mechanism for transporting policy values as part of interoperable object references and within requests.

**See Also**     [Messaging::RoutingPolicy](#)

# Messaging::ExceptionHolder Value Type

The messaging callback model uses an `ExceptionHolder` to deliver exceptions. Because the [ReplyHandler](#) implements an IDL interface, all arguments passed to its operations must be defined in IDL also. However, exceptions cannot be passed as arguments to operations, but are only raised as part of a reply. An `ExceptionHolder` value is created to encapsulate the identity and contents of the exception that might be raised. An instance of this `ExceptionHolder` is passed as the argument to the [ReplyHandler](#) operation that indicates an exception was raised by the target. In addition to its exception state, the `ExceptionHolder` also has operations that raise the returned exception, so the [ReplyHandler](#) implementation can have the returned exception re-raised within its own context.

AMI operations do not raise user exceptions. Rather, user exceptions are passed to the implemented type specific [ReplyHandler](#). If an AMI operation raises a system exception with a completion status of `COMPLETED_NO`, the request has not been made. This clearly distinguishes exceptions raised by the server (which are returned via the [ReplyHandler](#)) from the local exceptions that caused the AMI to fail.

The `ExceptionHolder` value class implementation is provided by the ORB. For each interface, a type specific `ExceptionHolder` value is generated by the IDL compiler. This `ExceptionHolder` is implemented by the ORB and passed to an application using the callback model when exception replies are returned from the target. See the *CORBA Programmer's Guide* for more on the generated value types and operations.

The code is as follows:

**Enhancement** The `ExceptionHolder` class is not compliant with the CORBA Messaging specification.

### ExceptionHolder::byte\_order()

Returns the byte order for the exception.

Sets the byte order for the exception.

### ExceptionHolder::\_downcast()

Returns a pointer to the `ExceptionHolder` type for a derived class. Each value type class provides `_downcast()` as a portable way for applications to cast down the C++ inheritance hierarchy.

#### Parameters

- `_val` Pointer to the value type class to be downcast.
- If the value type instance pointed to by the argument is an instance of the value type class being downcast to, a pointer to the downcast-to class type is returned.
  - If the value type instance pointed to by the argument is not an instance of the value type class being downcast to, a null pointer is returned.
  - If a null pointer is passed to `_downcast()`, it returns a null pointer.

This is especially required after an invocation of [\\_copy\\_value\(\)](#).

**Enhancement** Orbix enhancement.

**See Also** [CORBA::ValueBase::\\_copy\\_value\(\)](#)

### ExceptionHolder::ExceptionHolder() Constructors

Constructors for the `ExceptionHolder`.

**Enhancement** Orbix enhancement.

### ExceptionHolder::~~ExceptionHolder() Destructor

The destructor for the `ExceptionHolder`.

---

**Enhancement** Orbix enhancement.

### **ExceptionHandler::get\_exception()**

Returns the exception.

**See Also** [Messaging::ExceptionHandler::get\\_exception\\_with\\_list\(\)](#)

**Enhancement** Orbix enhancement.

### **ExceptionHandler::get\_exception\_with\_list()**

Returns a list of exceptions.

**Enhancement** Orbix enhancement.

**See Also** [Messaging::ExceptionHandler::get\\_exception\(\)](#)

### **ExceptionHandler::is\_system\_exception()**

### **ExceptionHandler::\_it\_demarshal\_value()**

---

**Note:** For internal use only.

---

### **ExceptionHandler::\_it\_get\_fw\_type\_id()**

---

**Note:** For internal use only.

---

**ExceptionHandler::\_it\_get\_safe\_bases()**

---

**Note:** For internal use only.

---

**ExceptionHandler::\_it\_marshall\_value()**

---

**Note:** For internal use only.

---

**ExceptionHandler::\_it\_type()**

---

**Note:** For internal use only.

---

**ExceptionHandler::\_local\_narrow()**

---

**Note:** For internal use only.

---

**ExceptionHandler::marshaled\_exception()**

**Enhancement** Orbix enhancement.

**ExceptionHandler::marshaled\_exception\_seq** Sequence

---

**Enhancement** Orbix enhancement.





# Messaging::RebindPolicy Class

The `RebindPolicy` is a client-side QoS policy that specifies whether or not the ORB is allowed to transparently relocate the target corresponding to an object reference. The default `RebindPolicy` supports this transparent rebind.

*Rebinding* means changing the client-visible QoS as a result of replacing the IOR profile used by a client's object reference with a new IOR profile. *Transparent rebinding* is when this happens without notice to the client application.

If your application has rigorous QoS requirements, transparent rebinding can cause problems. For instance, unexpected errors may occur if your application sets its QoS policies appropriately for an object reference, and then the ORB transparently changes the application's assumptions about that reference by obtaining a new IOR. Your applications can prevent the ORB from silently changing the IOR Profile and therefore the server-side QoS that you have assumed. A more rigorous value of this policy even precludes the ORB from silently closing and opening connections such as when IIOP is being used.

`RebindPolicy` is a local object derived from [CORBA::Policy](#).

...[See page 5](#) for descriptions of the standard helper methods:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

## `RebindPolicy::_local_narrow()`

---

**Note:** For internal use only.

---

### RebindPolicy::rebind\_mode()

Returns the effective rebind policy mode. The effective policies of other types for this object reference may change from invocation to invocation.

For GIOP-based protocols an object reference is considered bound once it is in a state where a locate-request message would result in a locate-reply message with status indicating where the object is. If `rebind_mode()` returns an effective policy value of [TRANSPARENT](#), the ORB will silently forward any subsequent messages.

Regardless of the rebind policy in effect, you can always explicitly requested rebind or reconnection by calling [Object::\\_validate\\_connection\(\)](#). When instances of [RebindPolicy](#) are created, a value of type [RebindMode](#) is passed to [ORB::create\\_policy\(\)](#).

### Exceptions

REBIND

Raised if:

- The effective policy value is [NO\\_REBIND](#) and if any rebind handling would cause a client-visible change in policies.
- The effective policy value is [NO\\_RECONNECT](#) and if any rebind handling would cause a client-visible change in policies, or if a new connection must be opened.

### See Also

[Messaging::RebindMode](#)  
[CORBA::ORB::create\\_policy\(\)](#)  
[CORBA::Object::\\_validate\\_connection\(\)](#)

### RebindPolicy::~~RebindPolicy() Destructor

The destructor for the object.

# Messaging::ReplyHandler Base Class

This is the base class for the messaging callback model. A `ReplyHandler` is a CORBA object, implemented by the client application, which encapsulates the functionality for handling an asynchronous reply. The `ReplyHandler` is used with an asynchronous method invocation (AMI). The `ReplyHandler` is passed to an AMI and it is invoked when the reply to that request is available.

In the callback model, the client passes a reference to a reply handler (a client side CORBA object implementation that handles the reply for a client request), in addition to the normal parameters needed by the request. The reply handler interface defines operations to receive the results of that request (including inout and out values and possible exceptions). The `ReplyHandler` is a normal CORBA object that is implemented by the programmer as with any object implementation.

You must write the implementation for a type-specific `ReplyHandler`. A client obtains an object reference for this `ReplyHandler` and passes it as part of the AMI. When the server completes the request, its reply is delivered as an invocation on the `ReplyHandler` object. This invocation is made on the `ReplyHandler` using the normal POA techniques of servant and object activation. As a result, the callback operation may be handled in a different programming context than that in which the original request was made.

Exceptions can only be raised as part of a reply in the callback model. You use an [ExceptionHolder](#) to handle these exception replies. You create an [ExceptionHolder](#) value to encapsulate the identity and contents of an exception that might be raised, and an instance of this [ExceptionHolder](#) is passed as the argument to the `ReplyHandler` operation to indicate if an exception was raised by the target.

For each operation in an interface, corresponding callback asynchronous method signatures are generated by the IDL compiler. See the *CORBA Programmer's Guide* for generated methods and how to write your asynchronous callback implementations.

[See page 5](#) for descriptions of the standard helper methods:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

### **ReplyHandler::\_local\_narrow()**

---

**Note:** For internal use only.

---

### **ReplyHandler::~~ReplyHandler() Destructor**

The destructor for the object.

# Messaging::RoutingPolicy Class

The `RoutingPolicy` is a QoS policy that specifies whether or not the ORB must ensure delivery of a request through the use of queueing. This interface is a local object derived from [CORBA::Policy](#).

When you create instances of `RoutingPolicy`, you pass a value of type [RoutingTypeRange](#) to [CORBA::ORB::create\\_policy\(\)](#). An instance of `RoutingPolicy` may be specified when creating a POA and therefore may be represented in object references.

In addition, a POA's `RoutingPolicy` is visible to clients through the object references it creates, and reconciled with the client's override. If set on both the client and server, reconciliation is performed by intersecting the server-specified `RoutingPolicy` range with the range of the client's effective override.

See [page 5](#) for descriptions of the standard helper methods:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

## `RoutingPolicy::_local_narrow()`

---

**Note:** For internal use only.

---

## `RoutingPolicy::~~RoutingPolicy()` Destructor

The destructor for the object.

**RoutingPolicy::routing\_range()**

Returns the routing type range.

# Messaging::SyncScopePolicy Class

The `SyncScopePolicy` is an ORB-level QoS policy that modifies the behavior of oneway operations. (Operations are specified in IDL with the `oneway` keyword.) This policy is only applicable as a client-side override. It is applied to oneway operations to indicate the synchronization scope with respect to the target of that operation request. It is ignored when any non-oneway operation is invoked. This policy is also applied when the DII is used with a flag of `INV_NO_RESPONSE` because the DII is not required to consult an interface definition to determine if an operation is declared oneway. The default value of this policy is not defined.

`SyncScopePolicy` is a local object derived from [CORBA::Policy](#). You create instances of `SyncScopePolicy` by passing a value of type [Messaging::SyncScope](#) to [CORBA::ORB::create\\_policy\(\)](#). The client's `SyncScopePolicy` is propagated within a request in the request header's response flags. Your applications must explicitly set a `SyncScopePolicy` to ensure portability across ORB implementations.

See [page 5](#) for descriptions of the standard helper methods:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

## `SyncScopePolicy::_local_narrow()`

---

**Note:** For internal use only.

---

### **SyncScopePolicy::synchronization()**

Returns the level of synchronization.

#### **See Also**

`Messaging::SyncScope`

### **SyncScopePolicy::~~SyncScopePolicy() Destructor**

The destructor for the object.



# OrbixEventsAdmin Module

The previous IONA implementation of the CORBA event service, `OrbixEvents`, provided the event channel administration interface, `ChannelManager`, defined in the module `OrbixEventsAdmin`, to allow Orbix 3.x clients to create and manipulate multiple event channels within an `OrbixEvents` server.

Orbix defines the `ChannelManager` interface for backwards compatibility with `OrbixEvents` users. This interface is defined in the file `orbixevents.idl` in the `include/idl` directory.

---

---

**WARNING:** The `orbixevents.idl` file is deprecated. All new clients using the event service should be using the interfaces provided in the `IT_EventChannelAdmin` module (defined in `event_channel_admin.idl`).

---

---

Existing clients can contact the event service by calling `resolve_initial_references("EventService")` and narrowing the reference from `OrbixEventsAdmin::ChannelManager`.



# OrbixEventsAdmin::ChannelManager

The previous IONA implementation of the CORBA event service, `OrbixEvents`, provided the event channel administration interface, `ChannelManager`, defined in the module `OrbixEventsAdmin`, to allow Orbix 3.x clients to create and manipulate multiple event channels within an `OrbixEvents` server.

Orbix defines the `ChannelManager` interface for backwards compatibility with `OrbixEvents` users. This interface is defined in the file `orbixevents.idl` in the `include/idl` directory.

---

---

**WARNING:** The `orbixevents.idl` file is deprecated. All new clients using the event service should be using the interfaces provided in the `IT_EventChannelAdmin` module (defined in `event_channel_admin.idl`).

---

---

Existing clients can contact the event service by calling `resolve_initial_references("EventService")` and narrowing the reference from `OrbixEventsAdmin::ChannelManager`.

## ChannelManager::create()

```
CosEventChannelAdmin::EventChannel create(in string channel_id)
raises(duplicateChannel);
```

Creates an event channel.

### Parameters

<code>channel_id</code>	The channel identifier for the event channel. The exception <code>duplicateChannel</code> is raised if the channel identifier specified in <code>channel_id</code> names an existing channel.  "Assigning Identifiers to Event Channels" on page 87 describes the format of channel identifiers.
-------------------------	--

---

## ChannelManager::find()

```
CosEventChannelAdmin::EventChannel find(in string channel_id)
raises (noSuchChannel);
```

Finds the event channel associated with the channel identifier `channel_id`.

### Parameters

<code>channel_id</code>	The channel identifier for the event channel. The exception <code>noSuchChannel</code> is raised if the channel identifier specified in <code>channel_id</code> does not exist.  “Assigning Identifiers to Event Channels” on page 87 describes the format of channel identifiers.
-------------------------	--

## ChannelManager::findByRef()

```
string findByRef(
    in CosEventChannelAdmin::EventChannel channel_ref)
raises (noSuchChannel);
```

Finds the channel identifier of the event channel specified in `channel_ref`.

### Parameters

<code>channel_ref</code>	The object reference for the event channel. If <code>channel_ref</code> does not exist within the event server, <code>findByRef()</code> raises the exception <code>noSuchChannel</code> .
--------------------------	--

## ChannelManager::list()

```
stringSeq list ();
```

Lists the generic event channels contained within the channel manager’s event server.

### ChannelManager::createTyped()

```
CosTypedEventChannelAdmin::TypedEventChannel createTyped(in string  
    channel_id)  
raises(duplicateChannel);
```

Creates a typed event channel.

#### Parameters

channel_id	The channel identifier for the typed event channel. The exception <code>duplicateChannel</code> is raised if the channel identifier specified in <code>channel_id</code> names an existing typed event channel.
------------	---

### ChannelManager::findTyped()

```
CosTypedEventChannelAdmin::TypedEventChannel findTyped(in string  
    channel_id)  
raises (noSuchChannel);
```

Finds the typed event channel associated with the channel identifier `channel_id`.

#### Parameters

channel_id	The channel identifier for the typed event channel. The exception <code>noSuchChannel</code> is raised if the channel identifier specified in <code>channel_id</code> does not exist.
------------	---

### ChannelManager::findTypedByRef()

```
string findTypedByRef(in CosTypedEventChannelAdmin::  
    TypedEventChannel channel_ref)  
raises (noSuchChannel);
```

Finds the channel identifier of the typed event channel specified in `channel_ref`.

---

## Parameters

`channel_ref` The object reference for the typed event channel. If `channel_ref` does not exist within the event server, `findByRef()` raises the exception `noSuchChannel`.

## **ChannelManager::listTyped()**

```
stringSeq listTyped();
```

Lists the typed event channels contained within the channel manager's event server.

## Unsupported Operations

The Application Server Platform event service does not support finding channels by reference. Therefore the `ChannelManager` reference will throw `NO_IMPLEMENT` for the following operations:

- `findByRef()`
- `findByTypedRef()`

# PortableInterceptor Module

The `PortableInterceptor` module consists of these interfaces:

[ClientRequestInfo](#)  
[ClientRequestInterceptor](#)  
[Current](#)  
[Interceptor](#)  
[IORInfo](#)  
[IORInterceptor](#)  
[ORBInitializer](#)  
[ORBInitInfo](#)  
[PolicyFactory](#)  
[RequestInfo](#)  
[ServerRequestInfo](#)  
[ServerRequestInterceptor](#)

The `PortableInterceptor` module also has the following exceptions and data types:

- [InvalidSlot](#) exception
- [ForwardRequest](#) exception
- [ReplyStatus](#) type
- [SlotId](#) type

## PortableInterceptor::ForwardRequest Exception

```
// IDL
exception ForwardRequest {
    Object forward;
    boolean permanent;
};
```

The `ForwardRequest` exception allows an `Interceptor` to indicate to the ORB that a retry of the request should occur with the new object given in the exception. The `permanent` flag indicates whether the `forward` object is to become permanent or used only on the forwarded request.

---

If an Interceptor raises a `ForwardRequest` exception, no other Interceptors are called for that interception point. The remaining Interceptors in the Flow Stack have their appropriate ending interception point called: `receive_other` on the client, or `send_other` on the server. The `reply_status` in the `receive_other` or `send_other` would be `LOCATION_FORWARD` or `LOCATION_FORWARD_PERMANENT`, depending on the value of the permanent element of `ForwardRequest`.

### PortableInterceptor::InvalidSlot Exception

```
// IDL
exception InvalidSlot {};
```

Raised when a slot ID does not match an allocated slot.

### PortableInterceptor::ReplyStatus Type

```
// IDL
typedef short ReplyStatus;
// Valid reply_status values:
const ReplyStatus SUCCESSFUL = 0;
const ReplyStatus SYSTEM_EXCEPTION = 1;
const ReplyStatus USER_EXCEPTION = 2;
const ReplyStatus LOCATION_FORWARD = 3;
const ReplyStatus LOCATION_FORWARD_PERMANENT = 4;
const ReplyStatus TRANSPORT_RETRY = 5;
```

This type is used to define an attribute describing the state of the result of an operation invocation.

#### See Also

[RequestInfo::reply\\_status](#)

### PortableInterceptor::SlotId Type

```
// IDL
typedef unsigned long SlotId;
```

This type is used to define a slot ID, identifying a slot within its table.



# PortableInterceptor:: ClientRequestInfo Interface

This is a locally constrained interface.

```
// IDL
local interface ClientRequestInfo : RequestInfo {
    readonly attribute Object target;
    readonly attribute Object effective\_target;
    readonly attribute IOP::TaggedProfile effective\_profile;
    readonly attribute any received\_exception;
    readonly attribute CORBA::RepositoryId received\_exception\_id;

    IOP::TaggedComponent get\_effective\_component(
        in IOP::ComponentId id
    );
    IOP_N::TaggedComponentSeq get\_effective\_components(
        in IOP::ComponentId id
    );
    CORBA::Policy get\_request\_policy(
        in CORBA::PolicyType type
    );
    void add\_request\_service\_context(
        in IOP::ServiceContext service_context,
        in boolean replace
    );
};
```

ClientRequestInfo is an object through which the client-side Interceptor can access request information. It is passed to the client-side interception points, just as ServerRequestInfo is passed to server-side interception points. As there is information that is common to both, they both inherit from a common interface—RequestInfo.

Some attributes and operations on `ClientRequestInfo` are not valid at all interception points. [Table 22](#) shows the validity of each attribute or operation. If it is not valid, attempting to access it will result in a `BAD_INV_ORDER` being raised with a standard minor code of 10.

**Table 22:** *ClientRequestInfo* Validity

	<b>send_request</b>	<b>send_poll</b>	<b>receive_reply</b>	<b>receive_exception</b>	<b>receive_other</b>
<i>request_id</i>	Yes	Yes	Yes	Yes	Yes
<i>operation</i>	Yes	Yes	Yes	Yes	Yes
<i>arguments</i>	Yes (note 1)	No	Yes	No	No
<i>exceptions</i>	Yes	No	Yes	Yes	Yes
<i>contexts</i>	Yes	No	Yes	Yes	Yes
<i>operation_context</i>	Yes	No	Yes	Yes	Yes
<i>result</i>	No	No	Yes	No	No
<i>response_expected</i>	Yes	Yes	Yes	Yes	Yes
<i>sync_scope</i>	Yes	No	Yes	Yes	Yes
<i>reply_statuses</i>	No	No	Yes	Yes	Yes
<i>forward_reference</i>	No	No	No	No	Yes (note 2)
<i>get_slot</i>	Yes	Yes	Yes	Yes	Yes
<i>get_request_service_context</i>	Yes	No	Yes	Yes	Yes
<i>get_reply_service_context</i>	No	No	Yes	Yes	Yes

**Table 22:** *ClientRequestInfo* Validity

	<b>send_request</b>	<b>send_poll</b>	<b>receive_reply</b>	<b>receive_exception</b>	<b>receive_other</b>
<i>target</i>	Yes	Yes	Yes	Yes	Yes
<i>effective_target</i>	Yes	Yes	Yes	Yes	Yes
<i>effective_profile</i>	Yes	Yes	Yes	Yes	Yes
<i>received_exception</i>	No	No	No	Yes	No
<i>received_exception_id</i>	No	No	No	Yes	No
<i>get_effective_component</i>	Yes	No	Yes	Yes	Yes
<i>get_effective_components</i>	Yes	No	Yes	Yes	Yes
<i>get_request_policy</i>	Yes	No	Yes	Yes	Yes
<i>add_request_service_context</i>	Yes	No	No	No	No

**Notes**

1. When *ClientRequestInfo* is passed to *send\_request*, there is an entry in the list for every argument, whether in, inout, or out. But only the in and inout arguments are available.

2. If the *reply\_status* attribute is not *LOCATION\_FORWARD* or *LOCATION\_FORWARD\_PERMANENT*, accessing this attribute raises *BAD\_INV\_ORDER* with a standard minor code of 10.

**See Also**

[ServerRequestInfo](#) : [RequestInfo](#); *RequestInfo*

---

## ClientRequestInfo::add\_request\_service\_context()

```
// IDL
void add_request_service_context(
    in IOP::ServiceContext service_context,
    in boolean replace
);
```

This operation allows Interceptors to add service contexts to a request for information. There is no declaration of the order of the service contexts. They may or may not appear in the order that they are added.

### Parameters

<code>service_context</code>	The <code>IOP::ServiceContext</code> to be added to the request.
<code>replace</code>	Indicates the behavior of this operation when a service context already exists with the given ID: <ul style="list-style-type: none"><li>• <code>true</code>: the existing service context is replaced by the new one.</li><li>• <code>false</code>: <code>BAD_INV_ORDER</code> with minor code of 11 is raised.</li></ul>

## ClientRequestInfo::effective\_profile Attribute

```
// IDL
readonly attribute IOP::TaggedProfile effective_profile;
```

This attribute is the profile that is used to send a request for information. If a location forward has occurred for this operation's object and that object's profile changed accordingly, then this profile is that located profile.

## ClientRequestInfo::effective\_target Attribute

```
// IDL
readonly attribute Object effective_target;
```

This attribute is the actual object on which a request for information is invoked. If the `reply_status` is `LOCATION_FORWARD`, then on subsequent requests, `effective_target` contains the forwarded IOR while `target`

---

remains unchanged. If the `reply_status` is `LOCATION_FORWARD_PERMANENT`, then on subsequent requests, both `effective_target` and `target` contains the forwarded IOR.

### **ClientRequestInfo::get\_effective\_component()**

```
// IDL
IOP::TaggedComponent get_effective_component(
    in IOP::ComponentId id
);
```

This operation returns the `IOP::TaggedComponent` with the given ID from the profile selected for this request. If there is more than one component for a given component ID, it is undefined which component this operation returns.

If there is more than one component for a given component ID, call `get_effective_components` instead.

#### **Parameters**

`id`                    The `IOP::ComponentId` of the component that is to be returned.

#### **Exceptions**

`BAD_PARAM,`            No component exists for the given component ID.  
minor code 25

### **ClientRequestInfo::get\_effective\_components()**

```
// IDL
IOP_N::TaggedComponentSeq get_effective_components(
    in IOP::ComponentId id
);
```

This operation returns all the tagged components with the given ID from the profile elected for this request. This sequence is in the form of an `IOP::TaggedComponentSeq`.

---

## Parameters

`id` The `IOP::ComponentId` of the components which are to be returned.

## Exceptions

`BAD_PARAM`,  
minor code 25 No component exists for the given component ID.

## ClientRequestInfo::get\_request\_policy()

```
// IDL
CORBA::Policy get_request_policy(
    in CORBA::PolicyType type
);
```

This operation returns the given policy in effect for the current request for information.

## Parameters

`type` The `CORBA::PolicyType` that specifies the policy to be returned.

## Exceptions

`INV_POLICY`,  
minor code 1 The policy type is not valid either because the specified type is not supported by this ORB or because a policy object of that type is not associated with this Object.

## ClientRequestInfo::received\_exception Attribute

```
// IDL
readonly attribute any received_exception;
```

This attribute is an `any` that contains the exception to be returned to the client.

If the exception is a user exception which cannot be inserted into an `any` (for example, it is unknown or the bindings do not provide the `TypeCode`, this

---

attribute will be an `any` containing the system exception `UNKNOWN` with a standard minor code of 1.

However, the `RepositoryId` of the exception is available in the `received_exception_id` attribute.

### **ClientRequestInfo::received\_exception\_id Attribute**

```
// IDL
readonly attribute CORBA::RepositoryId received_exception_id;
```

This attribute is the `CORBA::RepositoryId` of the exception to be returned to the client.

### **ClientRequestInfo::target Attribute**

```
// IDL
readonly attribute Object target;
```

This attribute is the object that the client called to perform the operation.





# PortableInterceptor:: ClientRequestInterceptor Interface

This is a locally constrained interface.

```
// IDL
local interface ClientRequestInterceptor : Interceptor {
    void send\_request(
        in ClientRequestInfo ri
    ) raises (ForwardRequest);
    void send\_poll(
        in ClientRequestInfo ri
    );
    void receive\_reply(
        in ClientRequestInfo ri
    );
    void receive\_exception(
        in ClientRequestInfo ri
    ) raises (ForwardRequest);
    void receive\_other(
        in ClientRequestInfo ri
    ) raises (ForwardRequest);
};
```

A request Interceptor is designed to intercept the flow of a request/reply sequence through the ORB at specific points so that services can query the request information and manipulate the service contexts which are propagated between clients and servers.

The primary use of request Interceptors is to enable ORB services to transfer context information between clients and servers. `ClientRequestInterceptor` provides the client-side request interceptor.

**See Also** [Interceptor](#)

---

## ClientRequestInterceptor::receive\_exception()

```
// IDL
void receive_exception(
    in ClientRequestInfo ri
) raises (ForwardRequest);
```

This interception point is called when an exception occurs. It allows an Interceptor to query the exception's information before it is raised to the client. This interception point can raise a system exception. This has the effect of changing the exception that successive Interceptors popped from the Flow Stack receive on their calls to `receive_exception`. The exception raised to the client is the last exception raised by an Interceptor, or the original exception if no Interceptor changes the exception.

This interception point can also raise a `ForwardRequest` exception (see [“PortableInterceptor::ForwardRequest Exception” on page 1105](#) for details on this exception). If an Interceptor raises this exception, no other Interceptors' `receive_exception` operations are called. The remaining Interceptors in the Flow Stack are popped and have their `receive_other` interception point called.

If the `completion_status` of the exception is not `COMPLETED_NO`, then it is inappropriate for this interception point to raise a `ForwardRequest` exception. The request's at-most-once semantics would be lost.

Compliant Interceptors that follow `completion_status` semantics raise a system exception from this interception point. If the original exception is a system exception, the `completion_status` of the new exception is the same as the original. If the original exception is a user exception, then the `completion_status` of the new exception is `COMPLETED_YES`.

Under some conditions, depending on what policies are in effect, an exception (such as `COMM_FAILURE`) can result in a retry of the request. While this retry is a new request with respect to Interceptors, there is one point of correlation between the original request and the retry: because control has not returned to the client, the `PortableInterceptor::Current` for both the original request and the retrying request is the same.

---

## ClientRequestInterceptor::receive\_other()

```
// IDL
void receive_other(
    in ClientRequestInfo ri
) raises (ForwardRequest);
```

This interception point allows an Interceptor to query the information available when a request results in something other than a normal reply or an exception.

For example, a request could result in a retry (for example, a GIOP Reply with a `LOCATION_FORWARD` status was received); or on asynchronous calls, the reply does not immediately follow the request, but control returns to the client and an ending interception point is called.

For retries, depending on the policies in effect, a new request may or may not follow when a retry has been indicated. If a new request does follow there is one point of correlation between the original request and the retry, with respect to Interceptors, and for as long as this request is a new request. This is because control has not returned to the client, and so the request scoped `PortableInterceptor::Current` for both the original request and the retrying request is the same.

This interception point can raise a system exception. If it does, no other Interceptors' `receive_other` operations are called. The remaining Interceptors in the Flow Stack are popped and have their `receive_exception` interception point called.

This interception point can also raise a `ForwardRequest` exception (see [“PortableInterceptor::ForwardRequest Exception” on page 1105](#) for details on this exception). If an Interceptor raises this exception, successive Interceptors' `receive_other` operations are called with the new information provided by the `ForwardRequest` exception.

Compliant Interceptors properly follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` must be `COMPLETED_NO`. If the target invocation had completed, this interception point would not be called.

---

## ClientRequestInterceptor::receive\_reply()

```
// IDL
void receive_reply(
    in ClientRequestInfo ri
);
```

This interception point allows an Interceptor to query the information on a reply, after it is returned from the server, and before control is returned to the client. This interception point can raise a system exception. If it does, no other Interceptors' `receive_reply` operations are called. The remaining Interceptors in the Flow Stack have their `receive_exception` interception point called.

Compliant Interceptors properly follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` is `COMPLETED_YES`.

## ClientRequestInterceptor::send\_poll()

```
// IDL
void send_poll(
    in ClientRequestInfo ri
);
```

This interception point allows an Interceptor to query information during a Time-Independent Invocation (TII) polling get reply sequence. With TII, an application can poll for a response to a request sent previously by the polling client or some other client. This poll is reported to Interceptors through the `send_poll` interception point and the response is returned through the `receive_reply` or `receive_exception` interception points. If the response is not available before the poll time-out expires, the system exception `TIMEOUT` is raised and `receive_exception` is called with this exception.

This interception point can raise a system exception. If it does, no other Interceptors' `send_poll` operations are called. Those Interceptors on the Flow Stack are popped and their `receive_exception` interception points are called. Compliant Interceptors properly follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` is `COMPLETED_NO`.

---

## ClientRequestInterceptor::send\_request()

```
// IDL
void send_request(
    in ClientRequestInfo ri
) raises (ForwardRequest);
```

This interception point allows an Interceptor to query request information and modify the service context before the request is sent to the server. This interception point can raise a system exception. If it does, no other Interceptors' `send_request` operations are called. Those Interceptors on the Flow Stack are popped and their `receive_exception` interception points are called.

This interception point may also raise a `ForwardRequest` exception (see [“PortableInterceptor::ForwardRequest Exception” on page 1105](#) for details of this exception). If an Interceptor raises this exception, no other Interceptors' `send_request` operations are called. Those Interceptors on the Flow Stack are popped and their `receive_other` interception points are called.

Compliant Interceptors follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` is `COMPLETED_NO`.



# PortableInterceptor::Current Interface

This is a locally constrained interface.

```
// IDL
local interface Current : CORBA::Current {
    any get\_slot(
        in SlotId id
    ) raises (InvalidSlot);
    void set\_slot(
        in SlotId id,
        in any data
    ) raises (InvalidSlot);
};
```

The `PortableInterceptor::Current` object (referred to as `PICurrent`) is a `Current` object that is used specifically by portable Interceptors to transfer thread context information to a request context. Portable Interceptors are not required to use `PICurrent`. But if information from a client's thread context is required at an Interceptor's interception points, then `PICurrent` can be used to propagate that information. `PICurrent` allows portable service code to be written regardless of an ORB's threading model.

On the client side, this information includes, but is not limited to, thread context information that is propagated to the server through a service context.

On the server side, this information includes, but is not limited to, service context information received from the client which is propagated to the target's thread context.

## **Current::get\_slot()**

```
// IDL
any get\_slot(
    in SlotId id
) raises (InvalidSlot);
```

---

A service can get the slot data it set in `PICurrent` with `get_slot()`. The return value is the data, in the form of an `any`, of the given slot identifier. If the given slot has not been set, an `any` containing a type code with a `TCKind` value of `tk_null` and no value is returned.

### Parameters

`id`                    The `SlotId` of the slot from which the data will be returned.

### Exceptions

`InvalidSlot`        `get_slot()` is called on a slot that has not been allocated.

### **Current::set\_slot()**

```
// IDL
void set_slot(
    in SlotId id,
    in any data
) raises (InvalidSlot);
```

A service sets data in a slot with `set_slot()`. The data is in the form of an `any`. If data already exists in that slot, it is overwritten.

### Parameters

`id`                    The `SlotId` of the slot to which the data is set.  
`data`                  The data, in the form of an `any`, which will be set to the identified slot.

### Exceptions

`InvalidSlot`        `set_slot()` is called on a slot that has not been allocated.



# PortableInterceptor::Interceptor Interface

This is a locally constrained interface.

```
// IDL
local interface Interceptor {
    readonly attribute string name;
};
```

Portable Interceptor interfaces and related type definitions reside in the module `PortableInterceptor`. All portable Interceptors inherit from the local interface `Interceptor`.

## Interceptor::name Attribute

```
// IDL
readonly attribute string name;
```

Each Interceptor can have a name that is used to order the lists of Interceptors. Only one Interceptor of a given name can be registered with the ORB for each Interceptor type. An Interceptor can be anonymous, that is, have an empty string as the name attribute. Any number of anonymous Interceptors can be registered with the ORB.



# PortableInterceptor::IORInfo Interface

This is a locally constrained interface.

```
// IDL
local interface IORInfo {
    CORBA::Policy get\_effective\_policy(
        in CORBA::PolicyType type
    );
    void add\_ior\_component(
        in IOP::TaggedComponent a_component
    );
    void add\_ior\_component\_to\_profile(
        in IOP::TaggedComponent a_component,
        in IOP::ProfileId profile_id
    );
};
```

In some cases, a portable ORB service implementation has to add information describing the server's or object's ORB service capabilities to object references. This permits the ORB service implementation in the client to function properly.

This is supported through the [IORInterceptor](#) and [IORInfo](#) interfaces. The IOR Interceptor is used to establish tagged components in the profiles within an IOR.

## IORInfo::add\_ior\_component()

```
// IDL
void add_ior_component(
    in IOP::TaggedComponent a_component
);
```

A portable ORB service implementation can call `add_ior_component` from its implementation of `establish_components` to add a tagged component to the set that is included when constructing IORs. The components in this set is included in all profiles.

---

Any number of components can exist with the same component ID.

### Parameters

`a_component`      The `IOP::TaggedComponent` to add.

### **.IORInfo::add\_ior\_component\_to\_profile()**

```
// IDL
void add_ior_component_to_profile(
    in IOP::TaggedComponent a_component,
    in IOP::ProfileId profile_id
);
```

A portable ORB service can call `add_ior_component_to_profile` from its implementation of `establish_components` to add a tagged component to the set that is included when constructing IORs. The components in this set included in the specified profile.

Any number of components can exist with the same component ID.

### Exceptions

`BAD_PARAM`,      The given profile ID does not define a known profile or it is  
minor code 26      impossible to add components to that profile.

### Parameters

`a_component`      The `IOP::TaggedComponent` to add.  
`profile_id`      The `IOP::ProfileId` of the profile to which this component  
is to be added.

### **.IORInfo::get\_effective\_policy()**

```
// IDL
CORBA::Policy get_effective_policy(
    in CORBA::PolicyType type
);
```

---

An ORB service implementation can determine what server side policy of a particular type is in effect for an IOR being constructed by calling `get_effective_policy()`. The returned `CORBA::Policy` object can only be a policy whose type was registered with `ORBInitInfo::register_policy_factory` (see [“ORBInitInfo::register\\_policy\\_factory\(\)” on page 1138](#)).

The return value is the effective `CORBA::Policy` object of the requested type.

### Parameters

<code>type</code>	The <code>CORBA::PolicyType</code> specifying the type of policy to return.
-------------------	---

### Exceptions

<code>INV_POLICY</code> , minor code 2	A policy for the given type was not registered with <code>register_policy_factory()</code> .
---	--



# PortableInterceptor::IORInterceptor Interface

This is a locally constrained interface.

```
// IDL
local interface IORInterceptor : Interceptor {
    void establish\_components(
        in IORInfo info
    );
};
```

In some cases, a portable ORB service implementation has to add information describing the server's or object's ORB service capabilities to object references. This permits the ORB service implementation in the client to function properly.

This is supported through the [IORInterceptor](#) and [IORInfo](#) interfaces. The IOR Interceptor is used to establish tagged components in the profiles within an IOR.

## IORInterceptor::establish\_components()

```
// IDL
void establish_components(
    in IORInfo info
);
```

A server side ORB calls `establish_components()` on all registered `IORInterceptor` instances when it is assembling the list of components that to be included in the profile or profiles of an object reference.

This operation is not necessarily called for each individual object reference. For example, the POA specifies policies at POA granularity and therefore, this operation might be called once per POA rather than once per object. In any case, `establish_components` is guaranteed to be called at least once for each distinct set of server policies.

---

An implementation of `establish_components` must not throw exceptions. If it does, the ORB ignores the exception and proceeds to call the next IOR Interceptor's `establish_components()` operation.

### Parameters

`info`            The `IORInfo` instance used by the ORB service to query applicable policies and add components to be included in the generated IORs.



# PortableInterceptor::ORBInitializer Interface

This is a locally constrained interface.

```
// IDL
local interface ORBInitializer {
    void pre\_init(
        in ORBInitInfo info
    );
    void post\_init(
        in ORBInitInfo info
    );
};
```

Interceptors are a means by which ORB services gain access to ORB processing, effectively becoming part of the ORB. Since Interceptors are part of the ORB, when `ORB_init` returns an ORB, the Interceptors have been registered.

Interceptors cannot be registered on an ORB after it has been returned by a call to `ORB_init`.

An Interceptor is registered by registering an associated `ORBInitializer` object that implements the `ORBInitializer` interface. When an ORB initializes, it calls each registered `ORBInitializer`, passing it an `ORBInitInfo` object that is used to register its Interceptor.

## **ORBInitializer::post\_init()**

```
// IDL
void post\_init(
    in ORBInitInfo info
);
```

This operation is called during ORB initialization. If a service must resolve initial references as part of its initialization, it can assume that all initial references are available at this point.

---

## Parameters

`info` This object provides initialization attributes and operations by which Interceptors can be registered.

## **ORBInitializer::pre\_init()**

```
// IDL
void pre_init(
    in ORBInitInfo info
);
```

This operation is called during ORB initialization. All calls to `ORBInitInfo::register_initial_reference` must be made at this point so that the list of initial references is complete for the `post_init` point.

## Parameters

`info` This object provides initialization attributes and operations by which Interceptors can be registered.

# PortableInterceptor::ORBInitInfo Interface

This is a locally constrained interface.

```
// IDL
local interface ORBInitInfo {
    typedef string ObjectId;
    exception DuplicateName {
        string name;
    };
    exception InvalidName {};
    readonly attribute CORBA::StringSeq arguments;
    readonly attribute string orb\_id;
    readonly attribute IOP_N::CodecFactory codec\_factory;

    void register\_initial\_reference(
        in ObjectId id,
        in Object obj
    ) raises (InvalidName);
    void resolve\_initial\_references(
        in ObjectId id
    ) raises (InvalidName);
    void add\_client\_request\_interceptor(
        in ClientRequestInterceptor interceptor
    ) raises (DuplicateName);
    void add\_server\_request\_interceptor(
        in ServerRequestInterceptor interceptor
    ) raises (DuplicateName);
    void add\_ior\_interceptor(
        in IORInterceptor interceptor
    ) raises (DuplicateName);
    SlotId allocate\_slot\_id();
    void register\_policy\_factory(
        in CORBA::PolicyType type,
        in PolicyFactory policy_factory
    );
};
```

---

```
    );  
};
```

Interceptors are a means by which ORB services gain access to ORB processing, effectively becoming part of the ORB. Since Interceptors are part of the ORB, when `ORB_init` returns an ORB, the Interceptors have been registered.

Interceptors cannot be registered on an ORB after it has been returned by a call to `ORB_init`.

An Interceptor is registered by registering an associated `ORBInitializer` object that implements the `ORBInitializer` interface. When an ORB initializes, it calls each registered `ORBInitializer`, passing it an `ORBInitInfo` object that is used to register its Interceptor.

### **ORBInitInfo::add\_client\_request\_interceptor()**

```
// IDL  
void add_client_request_interceptor(  
    in ClientRequestInterceptor interceptor  
    ) raises (DuplicateName);
```

This operation is used to add a client-side request Interceptor to the list of client-side request Interceptors.

#### **Parameters**

`interceptor`    The `ClientRequestInterceptor` to be added.

#### **Exceptions**

`DuplicateName`    A client-side request Interceptor has already been registered with this Interceptor's name.

### **ORBInitInfo::add\_ior\_interceptor()**

```
// IDL  
void add_ior_interceptor(  
    in IORInterceptor interceptor  
    ) raises (DuplicateName);
```

This operation is used to add an IOR Interceptor to the list of IOR Interceptors.

---

## Parameters

`interceptor` The `IORInterceptor` to be added.

## Exceptions

`DuplicateName` An IOR Interceptor has already been registered with this Interceptor's name.

### **ORBInitInfo:add\_server\_request\_interceptor()**

```
// IDL
void add_server_request_interceptor(
    in ServerRequestInterceptor interceptor
) raises (DuplicateName);
```

This operation is used to add a server-side request Interceptor to the list of server-side request Interceptors.

If a server-side request Interceptor has already been registered with this Interceptor's name, `DuplicateName` is raised.

## Parameters

`interceptor` The `ServerRequestInterceptor` to be added.

### **ORBInitInfo::allocate\_slot\_id()**

```
// IDL
SlotId allocate_slot_id();
```

A service calls `allocate_slot_id` to allocate a slot on `PortableInterceptor::Current`.

The return value is the allocated slot index.

### **ORBInitInfo::arguments Attribute**

```
// IDL
readonly attribute CORBA::StringSeq arguments;
```

---

This attribute contains the arguments passed to `ORB_init`. They may or may not contain the ORB's arguments.

### **ORBInitInfo::codec\_factory Attribute**

```
// IDL
readonly attribute IOP_N::CodecFactory codec_factory;
```

This attribute is the `IOP::CodecFactory`. The `CodecFactory` is normally obtained with a call to `ORB::resolve_initial_references` ("`CodecFactory`"), but as the ORB is not yet available and Interceptors, particularly when processing service contexts, require a `Codec`, a means of obtaining a `Codec` is necessary during ORB initialization.

### **ORBInitInfo::DuplicateName Exception**

```
// IDL
exception DuplicateName {
    string name;
};
```

Only one Interceptor of a given name can be registered with the ORB for each Interceptor type. If an attempt is made to register a second Interceptor with the same name, `DuplicateName` is raised.

An Interceptor can be anonymous, that is, have an empty string as the name attribute.

Any number of anonymous Interceptors may be registered with the ORB so, if the Interceptor being registered is anonymous, the registration operation will not raise `DuplicateName`.

### **ORBInitInfo::InvalidName Exception**

```
// IDL
exception InvalidName {};
```

This exception is raised by `register_initial_reference` and `resolve_initial_references`.

---

`register_initial_reference` raises `InvalidName` if this operation is called with an empty string `id`; or this operation is called with an `id` that is already registered, including the default names defined by OMG.

`resolve_initial_references` raises `InvalidName` if the name to be resolved is invalid.

## ORBInitInfo::ObjectId Type

```
// IDL
typedef string ObjectId;
```

See Also

[ORBInitInfo::register\\_initial\\_reference\(\)](#)

## ORBInitInfo::orb\_id Attribute

```
// IDL
readonly attribute string orb_id;
```

This attribute is the ID of the ORB being initialized.

## ORBInitInfo::register\_initial\_reference()

```
// IDL
void register_initial_reference(
    in ObjectId id,
    in Object obj
) raises (InvalidName);
```

If this operation is called with an `id`, “Y”, and an object, YY, then a subsequent call to `ORB::resolve_initial_references (“Y”)` will return object YY.

### Parameters

<code>id</code>	The ID by which the initial reference will be known.
<code>obj</code>	The initial reference itself.

---

## Exceptions

- `BAD_PARAM`, minor code 24 The `Object` parameter is null.
- `InvalidName` Raised if this operation is called with:
- an empty string id.
  - an id that is already registered, including the default names defined by OMG.

## Notes

This method is identical to an operation is available in the ORB interface. This same functionality exists here because the ORB, not yet fully initialized, is not yet available but initial references may need to be registered as part of Interceptor registration. The only difference is that the version of this operation on the ORB uses PIDL (`CORBA::ORB::ObjectId` and `CORBA::ORB::InvalidName`) whereas the version in this interface uses IDL defined in this interface; the semantics are identical.

## **ORBInitInfo::register\_policy\_factory()**

```
// IDL
void register_policy_factory(
    in CORBA::PolicyType type,
    in PolicyFactory policy_factory
);
```

Register a `PolicyFactory` for the given `PolicyType`.

## Parameters

`type` The `CORBA::PolicyType` that the given `PolicyFactory` serves.

`policy_factory` The factory for the given `CORBA::PolicyType`.

## Exceptions

`BAD_INV_ORDER` A `PolicyFactory` already exists for the given `PolicyType`.  
with minor code  
12



---

## ORBInitInfo::resolve\_initial\_references()

```
// IDL
void resolve_initial_references(
    in ObjectId id
) raises (InvalidName);
```

This operation is only valid during `post_init`. It is identical to `ORB::resolve_initial_references`. This same functionality exists here because the ORB, not yet fully initialized, is not yet available but initial references can be required from the ORB as part of Interceptor registration. The only difference is that the version of this operation on the ORB uses PIDL (`CORBA::ORB::ObjectId` and `CORBA::ORB::InvalidName`) whereas the version in this interface uses IDL defined in this interface; the semantics are identical.



# PortableInterceptor::PolicyFactory Interface

This is a locally constrained interface.

```
// IDL
local interface PolicyFactory {
    CORBA::Policy create_policy(
        in CORBA::PolicyType type,
        in any value
    ) raises (CORBA::PolicyError);
};
```

A portable ORB service implementation registers an instance of the `PolicyFactory` interface during ORB initialization in order to enable its policy types to be constructed using `CORBA::ORB::create_policy`. The POA is required to preserve any policy which is registered with `ORBInitInfo` in this manner.

## **PolicyFactory::create\_policy()**

```
// IDL
CORBA::Policy create_policy(
    in CORBA::PolicyType type,
    in any value
) raises (CORBA::PolicyError);
```

The ORB calls `create_policy` on a registered `PolicyFactory` instance when `CORBA::ORB::create_policy` is called for the `PolicyType` under which the `PolicyFactory` has been registered.

`create_policy` returns an instance of the appropriate interface derived from `CORBA::Policy` whose value corresponds to the specified `any`. If it cannot, it raises an exception as described for `CORBA::ORB::create_policy`.

---

## Parameters

type	A <code>CORBA::PolicyType</code> specifying the type of policy being created.
value	An any containing data with which to construct the <code>CORBA::Policy</code> .

# PortableInterceptor::RequestInfo Interface

This is a locally constrained interface.

```
// IDL
local interface RequestInfo {
    readonly attribute unsigned long request\_id;
    readonly attribute string operation;
    readonly attribute Dynamic::ParameterList arguments;
    readonly attribute Dynamic::ExceptionList exceptions;
    readonly attribute Dynamic::ContextList contexts;
    readonly attribute Dynamic::RequestContext operation\_context;
    readonly attribute any result;
    readonly attribute boolean response\_expected;
    readonly attribute Messaging::SyncScope sync\_scope;
    readonly attribute ReplyStatus reply\_status;
    readonly attribute Object forward\_reference;
    any get\_slot(
        in SlotId id
    ) raises (InvalidSlot);
    IOP::ServiceContext get\_request\_service\_context(
        in IOP::ServiceId id
    );
    IOP::ServiceContext get\_reply\_service\_context(
        in IOP::ServiceId id
    );
};
```

Each interception point is given an object through which the Interceptor can access request information. Client-side and server-side interception points are concerned with different information, so there are two information objects. `ClientRequestInfo` is passed to the client-side interception points and `ServerRequestInfo` is passed to the server-side interception points. But as there is information that is common to both, so they both inherit from a common interface: [RequestInfo](#).

---

**See Also**     [ClientRequestInfo](#); [ServerRequestInfo](#)

## RequestInfo::arguments Attribute

```
// IDL
readonly attribute Dynamic::ParameterList arguments;
```

This attribute is a `Dynamic::ParameterList` containing the arguments on the operation being invoked. If there are no arguments, this attribute is a zero length sequence.

### Exceptions

`NO_RESOURCES`, The environment does not provide access to the arguments—minor code 1     for example, in the case of Java portable bindings.

## RequestInfo::contexts Attribute

```
// IDL
readonly attribute Dynamic::ContextList contexts;
```

This attribute is a `Dynamic::ContextList` describing the contexts that can be passed on this operation invocation. If there are no contexts, this attribute is a zero length sequence.

### Exceptions

`NO_RESOURCES`, The environment does not provide access to the context list—minor code 1     for example, in the case of Java portable bindings.

## RequestInfo::exceptions Attribute

```
// IDL
readonly attribute Dynamic::ExceptionList exceptions;
```

This attribute is a `Dynamic::ExceptionList` describing the `TypeCodes` of the user exceptions that this operation invocation can raise. If there are no user exceptions, this attribute is a zero length sequence.

---

## Exceptions

`NO_RESOURCES`, The environment does not provide access to the exception  
minor code 1 list—for example, in the case of Java portable bindings.

## RequestInfo::forward\_reference Attribute

```
// IDL  
readonly attribute Object forward_reference;
```

If the `reply_status` attribute is `LOCATION_FORWARD` or `LOCATION_FORWARD_PERMANENT`, then this attribute contains the object to which the request is to be forwarded. It is indeterminate whether a forwarded request actually occurs.

## RequestInfo::get\_reply\_service\_context()

```
// IDL  
IOP::ServiceContext get_reply_service_context(  
    in IOP::ServiceId id  
);
```

This operation returns a copy of the service context with the given ID that is associated with the reply.

The return value is the `IOP::ServiceContext` obtained with the given identifier.

## Parameters

`id` The `IOP::ServiceId` of the service context which is to be returned.

## Exceptions

`BAD_PARAM` with The request's service context does not contain an entry for the  
minor code 23 specified ID.

---

## RequestInfo::get\_request\_service\_context()

```
// IDL
IOP::ServiceContext get_request_service_context(
    in IOP::ServiceId id
);
```

This operation returns a copy of the service context with the given ID that is associated with the request.

The return value is the `IOP::ServiceContext` obtained with the given identifier.

### Parameters

`id`                    The `IOP::ServiceId` of the service context which is to be returned.

### Exceptions

`BAD_PARAM` with    The request's service context does not contain an entry for the  
minor code 23    specified ID.

## RequestInfo::get\_slot()

```
// IDL
any get_slot(
    in SlotId id
) raises (InvalidSlot);
```

This operation returns the data from the given slot of the `PortableInterceptor::Current` that is in the scope of the request. If the given slot has not been set, then an `any` containing a type code with a `TCKind` value of `tk_null` is returned.

The return value is the slot data, in the form of an `any`, obtained with the given identifier.

### Parameters

`id`                    The `slotId` of the slot that is to be returned.

### Exceptions

`InvalidSlot`        The ID does not define an allocated slot.



---

## RequestInfo::operation Attribute

```
// IDL
readonly attribute string operation;
```

This attribute is the name of the operation being invoked.

## RequestInfo::operation\_context Attribute

```
// IDL
readonly attribute Dynamic::RequestContext operation_context;
```

This attribute is a `Dynamic::RequestContext` containing the contexts being sent on the request

## Exceptions

`NO_RESOURCES`, The environment does not provide access to the context—for minor code 1 example, in the case of Java portable bindings.

## RequestInfo::reply\_status Attribute

```
// IDL
readonly attribute ReplyStatus reply_status;
```

This attribute describes the state of the result of the operation invocation. Its value can be one of the following:

```
PortableInterceptor::SUCCESSFUL
PortableInterceptor::SYSTEM_EXCEPTION
PortableInterceptor::USER_EXCEPTION
PortableInterceptor::LOCATION_FORWARD
PortableInterceptor::LOCATION_FORWARD_PERMANENT
PortableInterceptor::TRANSPORT_RETRY
```

On the client:

- Within the `receive_reply` interception point, this attribute is only `SUCCESSFUL`.
- Within the `receive_exception` interception point, this attribute is either `SYSTEM_EXCEPTION` or `USER_EXCEPTION`.

- 
- Within the `receive_other` interception point, this attribute is any of `SUCCESSFUL`, `LOCATION_FORWARD`, `LOCATION_FORWARD_PERMANENT`, or `TRANSPORT_RETRY`.

`SUCCESSFUL` means an asynchronous request returned successfully.

`LOCATION_FORWARD` and `LOCATION_FORWARD_PERMANENT` mean that a reply came back with one of these as its status.

`TRANSPORT_RETRY` means that the transport mechanism indicated a retry: a GIOP reply with a status of `NEEDS_ADDRESSING_MODE`, for instance.

On the server:

- Within the `send_reply` interception point, this attribute is only `SUCCESSFUL`.
- Within the `send_exception` interception point, this attribute is either `SYSTEM_EXCEPTION` or `USER_EXCEPTION`.
- Within the `send_other` interception point, this attribute is any of: `SUCCESSFUL`, `LOCATION_FORWARD`, or `LOCATION_FORWARD_PERMANENT`.  
`SUCCESSFUL` means an asynchronous request returned successfully.  
`LOCATION_FORWARD` and `LOCATION_FORWARD_PERMANENT` mean that a reply came back with one of these as its status.

## RequestInfo::request\_id Attribute

```
// IDL
readonly attribute unsigned long request_id;
```

This ID uniquely identifies an active request/reply sequence. Once a request/reply sequence is concluded this ID may be reused.

Note that this id is not the same as the GIOP `request_id`. If GIOP is the transport mechanism used, then these IDs may very well be the same, but this is not guaranteed nor required.

## RequestInfo::response\_expected Attribute

```
// IDL
readonly attribute boolean response_expected;
```

---

This boolean attribute indicates whether a response is expected. On the client:

- A reply is not returned when `response_expected` is false, so `receive_reply` cannot be called.
- If an exception occurs, `receive_exception` is called.
- Otherwise `receive_other` is called.

On the client, within `send_poll`, this attribute is `true`.

### RequestInfo::result Attribute

```
// IDL
readonly attribute any result;
```

This attribute is an `any` containing the result of the operation invocation.

If the operation return type is `void`, this attribute is an `any` containing a type code with a `TKind` value of `tk_void` and no value.

### Exceptions

`NO_RESOURCES`, The environment does not provide access to the result—for minor code 1 example, in the case of Java portable bindings.

### RequestInfo::sync\_scope Attribute

```
// IDL
readonly attribute Messaging::SyncScope sync_scope;
```

This attribute, defined in the Messaging specification, is pertinent only when `response_expected` is false. If `response_expected` is true, the value of `sync_scope` is undefined. It defines how far the request progresses before control is returned to the client. This attribute may have one of the following values:

```
Messaging::SYNC_NONE
Messaging::SYNC_WITH_TRANSPORT
Messaging::SYNC_WITH_SERVER
Messaging::SYNC_WITH_TARGET
```

---

On the server, for all scopes, a reply is created from the return of the target operation call, but the reply does not return to the client. Although it does not return to the client, it does occur, so the normal server-side interception points are followed (that is, `receive_request_service_contexts`, `receive_request`, `send_reply` or `send_exception`). For `SYNC_WITH_SERVER` and `SYNC_WITH_TARGET`, the server does send an empty reply back to the client before the target is invoked. This reply is not intercepted by server-side Interceptors.

# PortableInterceptor:: ServerRequestInfo Interface

This is a locally constrained interface.

```
// IDL
local interface ServerRequestInfo : RequestInfo {
    readonly attribute any sending\_exception;
    readonly attribute CORBA::OctetSeq object\_id;
    readonly attribute CORBA::OctetSeq adapter\_id;
    readonly attribute
        CORBA::RepositoryId target\_most\_derived\_interface;
    CORBA::Policy get\_server\_policy(
        in CORBA::PolicyType type
    );
    void set\_slot(
        in SlotId id,
        in any data
    ) raises (InvalidSlot);
    boolean target\_is\_a(
        in CORBA::RepositoryId id
    );
    void add\_reply\_service\_context(
        in IOP::ServiceContext service_context,
        in boolean replace
    );
};
```

`ServerRequestInfo` is an object through which the server-side Interceptor can access request information. It is passed to the server-side interception points, just as `ClientRequestInfo` is passed to client-side interception points. As there is information that is common to both, they both inherit from a common interface—`RequestInfo`.

---

## ServerRequestInfo::adapter\_id Attribute

```
// IDL
readonly attribute CORBA::OctetSeq adapter_id;
```

This attribute is the opaque identifier for the object adapter.

## ServerRequestInfo::add\_reply\_service\_context()

```
// IDL
void add_reply_service_context(
    in IOP::ServiceContext service_context,
    in boolean replace
);
```

This operation allows Interceptors to add service contexts to the request. There is no declaration of the order of the service contexts. They may or may not appear in the order that they are added.

### Parameters

`service_context` The `IOP::ServiceContext` to add to the reply.

`replace` Indicates the behavior of this operation when a service context already exists with the given ID. If `false`, then `BAD_INV_ORDER` with a standard minor code of 11 is raised. If `true`, then the existing service context is replaced by the new one.

## ServerRequestInfo::get\_server\_policy()

```
// IDL
CORBA::Policy get_server_policy(
    in CORBA::PolicyType type
);
```

This operation returns the policy in effect for this operation for the given policy type. The returned `CORBA::Policy` object is a policy whose type was registered using `register_policy_factory`

---

## Parameters

`type` The `CORBA::PolicyType` which specifies the policy to be returned.

## Exceptions

`INV_POLICY`, minor code 2 A policy for the given type was not registered using `register_policy_factory()`.

## ServerRequestInfo::object\_id Attribute

```
// IDL
readonly attribute CORBA::OctetSeq object_id;
```

This attribute is the opaque `object_id` describing the target of the operation invocation.

## ServerRequestInfo::sending\_exception Attribute

```
// IDL
readonly attribute any sending_exception;
```

This attribute is an `any` that contains the exception to be returned to the client.

If the exception is a user exception which cannot be inserted into an `any` (that is, it is unknown or the bindings do not provide the `TypeCode`), this attribute is an `any` containing the system exception `UNKNOWN` with a standard minor code of 1.

## ServerRequestInfo::set\_slot()

```
// IDL
void set_slot(
    in SlotId id,
    in any data
) raises (InvalidSlot);
```

---

This operation allows an `Interceptor` to set a slot in the `PortableInterceptor::Current` that is in the scope of the request. If data already exists in that slot, it is overwritten.

### Parameters

`id`                    The `slotId` of the slot.  
`data`                   The data, in the form of an `any`, to store in that slot.

### Exceptions

`InvalidSlot`        The ID does not define an allocated slot.

## ServerRequestInfo::target\_is\_a()

```
// IDL
boolean target_is_a(
    in CORBA::RepositoryId id
);
```

This operation returns `true` if the servant is the given `RepositoryId`, and `false` if it is not.

### Parameters

`id`                    The caller wants to know if the servant is this `CORBA::RepositoryID`.

## ServerRequestInfo::target\_most\_derived\_interface Attribute

```
// IDL
readonly attribute
    CORBA::RepositoryId target_most_derived_interface;
```

This attribute is the `RepositoryID` for the most derived interface of the servant.



# PortableInterceptor:: ServerRequestInterceptor Interface

This is a locally constrained interface.

```
// IDL
local interface ServerRequestInterceptor : Interceptor {
    void receive\_request\_service\_contexts(
        in ServerRequestInfo ri
    ) raises (ForwardRequest);
    void receive\_request(
        in ServerRequestInfo ri
    ) raises (ForwardRequest);
    void send\_reply(
        in ServerRequestInfo ri
    );
    void send\_exception(
        in ServerRequestInfo ri
    ) raises (ForwardRequest);
    void send\_other(
        in ServerRequestInfo ri
    ) raises (ForwardRequest);
};
```

A request Interceptor is designed to intercept the flow of a request/reply sequence through the ORB at specific points so that services can query the request information and manipulate the service contexts which are propagated between clients and servers.

The primary use of request Interceptors is to enable ORB services to transfer context information between clients and servers. `ServerRequestInterceptor` provides the server-side request interceptor.

---

## ServerRequestInterceptor::receive\_request()

```
// IDL
void receive_request(
    in ServerRequestInfo ri
) raises (ForwardRequest);
```

This interception point allows an Interceptor to query request information after all the information, including operation parameters, is available. This interception point may or may not execute in the same thread as the target invocation.

In the DSI model, as the parameters are first available when the user code calls `arguments`, `receive_request` is called from within `arguments`. It is possible that `arguments` is not called in the DSI model. The target can call `set_exception` before calling `arguments`.

The ORB guarantees that `receive_request` is called once, either through `arguments` or through `set_exception`. If it is called through `set_exception`, requesting the arguments results in `NO_RESOURCES` being raised with a standard minor code of 1. This interception point can raise a system exception. If it does, no other Interceptors' `receive_request` operations are called. Those Interceptors on the Flow Stack are popped and their `send_exception` interception points are called.

This interception point can also raise a `ForwardRequest` exception. If an Interceptor raises this exception, no other Interceptors' `receive_request` operations are called. Those Interceptors on the Flow Stack are popped and their `send_other` interception points are called.

Compliant Interceptors follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` shall be `COMPLETED_NO`.

## ServerRequestInterceptor::receive\_request\_service\_contexts()

```
// IDL
void receive_request_service_contexts(
    in ServerRequestInfo ri
) raises (ForwardRequest);
```

---

At this interception point, Interceptors must get their service context information from the incoming request transfer it to `PortableInterceptor::Current` slots. This interception point is called before the servant manager is called. Operation parameters are not yet available at this point. This interception point may or may not execute in the same thread as the target invocation.

This interception point can raise a system exception. If it does, no other Interceptors' `receive_request_service_contexts` operations are called. Those Interceptors on the Flow Stack are popped and their `send_exception` interception points are called.

This interception point can also raise a `ForwardRequest` exception (see [“PortableInterceptor::ForwardRequest Exception” on page 1105](#)). If an Interceptor raises this exception, no other Interceptors' `receive_request_service_contexts` operations are called. Those Interceptors on the Flow Stack are popped and their `send_other` interception points are called.

Compliant Interceptors follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` is `COMPLETED_NO`.

## **ServerRequestInterceptor::send\_exception()**

```
// IDL
void send_exception(
    in ServerRequestInfo ri
) raises (ForwardRequest);
```

This interception point is called when an exception occurs. It allows an Interceptor to query the exception information and modify the reply service context before the exception is raised to the client.

This interception point can raise a system exception. This has the effect of changing the exception that successive Interceptors popped from the Flow Stack receive on their calls to `send_exception`. The exception raised to the client is the last exception raised by an Interceptor, or the original exception if no Interceptor changes the exception.

---

This interception point also raises a `ForwardRequest` exception (see [“PortableInterceptor::ForwardRequest Exception” on page 1105](#)). If an Interceptor raises this exception, no other Interceptors’ `send_exception` operations are called. The remaining Interceptors in the Flow Stack have their `send_other` interception points called.

If the `completion_status` of the exception is not `COMPLETED_NO`, then it is inappropriate for this interception point to raise a `ForwardRequest` exception. The request’s at-most-once semantics would be lost.

Compliant Interceptors follow `completion_status` semantics if they raise a system exception from this interception point. If the original exception is a system exception, the `completion_status` of the new exception is the same as on the original. If the original exception is a user exception, then the `completion_status` of the new exception shall be `COMPLETED_YES`.

## ServerRequestInterceptor::send\_other()

```
// IDL
void send_other(
    in ServerRequestInfo ri
) raises (ForwardRequest);
```

This interception point allows an Interceptor to query the information available when a request results in something other than a normal reply or an exception. For example, a request could result in a retry (for example, a GIOP Reply with a `LOCATION_FORWARD` status was received).

This interception point can raise a system exception. If it does, no other Interceptors’ `send_other` operations are called. The remaining Interceptors in the Flow Stack have their `send_exception` interception points called.

This interception point can also raise a `ForwardRequest` exception (see [“PortableInterceptor::ForwardRequest Exception” on page 1105](#)). If an Interceptor raises this exception, successive Interceptors’ operations are called with the new information provided by the `ForwardRequest` exception.

Compliant Interceptors follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` is `COMPLETED_NO`.

---

## ServerRequestInterceptor::send\_reply()

```
// IDL
void send_reply(
    in ServerRequestInfo ri
);
```

This interception point allows an Interceptor to query reply information and modify the reply service context after the target operation has been invoked and before the reply is returned to the client.

This interception point can raise a system exception. If it does, no other Interceptors' `send_reply` operations are called. The remaining Interceptors in the Flow Stack have their `send_exception` interception point called.

Compliant Interceptors follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` is `COMPLETED_YES`.



# PortableServer Overview

The `PortableServer` module includes a number of data structures and classes that are specific to a portable object adapter, or POA. This chapter describes the following:

- [“PortableServer Data Types, Constants, and Exceptions”](#)

The rest of the `PortableServer` classes and interfaces are described in subsequent chapters as follows:

- [“PortableServer::AdapterActivator Interface”](#)
- [“PortableServer::Current Interface”](#)
- [“PortableServer::DynamicImplementation Class”](#)
- [“PortableServer::POA Interface”](#)
- [“PortableServer::POAManager Interface”](#)
- [“PortableServer::ServantActivator Interface”](#)
- [“PortableServer::ServantLocator Interface”](#)
- [“PortableServer::ServantManager Interface”](#)

The `PortableServer` policy classes are described in subsequent chapters as follows:

- [“PortableServer::IdAssignmentPolicy Interface”](#)
- [“PortableServer::IdUniquenessPolicy Interface”](#)
- [“PortableServer::ImplicitActivationPolicy Interface”](#)
- [“PortableServer::LifespanPolicy Interface”](#)
- [“PortableServer::RequestProcessingPolicy Interface”](#)
- [“PortableServer::ServantRetentionPolicy Interface”](#)
- [“PortableServer::ThreadPolicy Interface”](#)

---

# PortableServer Data Types, Constants, and Exceptions

The `PortableServer` module contains the following common exception and data types:

**Table 23:** *PortableServer Common Types*

Common Types and Exceptions	Policy Value Enumerations	Policy ID Constants
<a href="#">ForwardRequest</a>	<a href="#">IdAssignmentPolicyValue</a>	<a href="#">ID_ASSIGNMENT_POLICY_ID</a>
<a href="#">Servant</a>	<a href="#">IdUniquenessPolicyValue</a>	<a href="#">ID_UNIQUENESS_POLICY_ID</a>
	<a href="#">ImplicitActivationPolicyValue</a>	<a href="#">IMPLICIT_ACTIVATION_POLICY_ID</a>
	<a href="#">LifespanPolicyValue</a>	<a href="#">LIFESPAN_POLICY_ID</a>
	<a href="#">RequestProcessingPolicyValue</a>	<a href="#">REQUEST_PROCESSING_POLICY_ID</a>
	<a href="#">ServantRetentionPolicyValue</a>	<a href="#">SERVANT_RETENTION_POLICY_ID</a>
	<a href="#">ThreadPolicyValue</a>	<a href="#">THREAD_POLICY_ID</a>

## PortableServer::ForwardRequest Exception

```
//IDL
exception ForwardRequest {
    Object forward_reference;
};
```

In addition to standard CORBA exceptions, a servant manager is capable of raising a `ForwardRequest` exception. This exception includes an object reference.

## PortableServer::ID\_ASSIGNMENT\_POLICY\_ID Constant

```
//IDL
const CORBA::PolicyType ID_ASSIGNMENT_POLICY_ID = 19;
```

Defines an ID for the policy `IdAssignmentPolicy`.



### PortableServer::IdAssignmentPolicyValue Enumeration

```
// IDL
enum IdAssignmentPolicyValue {
    USER_ID,
    SYSTEM_ID
};
```

One of the following values can be supplied when creating an `IdAssignmentPolicy` policy.

<code>USER_ID</code>	Objects created with the POA are assigned an <code>ObjectId</code> only by the application.
<code>SYSTEM_ID</code>	Objects created with the POA are assigned an <code>ObjectId</code> only by the POA. If the POA also has the <code>PERSISTENT</code> policy for its objects, the assigned <code>ObjectId</code> must be unique across all instantiations of the same POA.

### PortableServer::ID\_UNIQUENESS\_POLICY\_ID Constant

```
//IDL
const CORBA::PolicyType ID_UNIQUENESS_POLICY_ID = 18;
```

Defines an ID for the policy `IdUniquenessPolicy`.

### PortableServer::IdUniquenessPolicyValue Enumeration

```
// IDL
enum IdUniquenessPolicyValue {
    UNIQUE_ID,
    MULTIPLE_ID
};
```

One of the following values can be supplied when creating an `IdUniquenessPolicy` policy.

<code>UNIQUE_ID</code>	Servants activated with the POA support exactly one <code>ObjectId</code> .
------------------------	---

---

MULTIPLE\_ID

A servant activated with the POA may support one or more `ObjectId` types.

### **PortableServer::IMPLICIT\_ACTIVATION\_POLICY\_ID Constant**

```
//IDL
const CORBA::PolicyType IMPLICIT_ACTIVATION_POLICY_ID = 20;
```

Defines an ID for the policy `ImplicitActivationPolicy`.

### **PortableServer::ImplicitActivationPolicyValue Enumeration**

```
// IDL
enum ImplicitActivationPolicyValue {
    IMPLICIT_ACTIVATION,
    NO_IMPLICIT_ACTIVATION
};
```

One of the following values can be supplied when creating an `ImplicitActivationPolicy` policy.

`IMPLICIT_ACTIVATION` The POA will support implicit activation of servants. `IMPLICIT_ACTIVATION` also requires the `SYSTEM_ID` and `RETAIN` policy values.

`NO_IMPLICIT_ACTIVATION` The POA will not support implicit activation of servants.

### **PortableServer::LIFESPAN\_POLICY\_ID Constant**

```
//IDL
const CORBA::PolicyType LIFESPAN_POLICY_ID = 17;
```

Defines an ID for the policy `LifeSpanPolicy`.

### **PortableServer::LifespanPolicyValue Enumeration**

```
// IDL
```

```
enum LifespanPolicyValue {
    TRANSIENT,
    PERSISTENT
};
```

One of the following values can be supplied when creating a `LifespanPolicy` policy:

TRANSIENT	The objects implemented in the POA cannot outlive the POA instance in which they are first created. Once the POA is deactivated, use of any object references generated from it will result in an <code>OBJECT_NOT_EXIST</code> exception.
PERSISTENT	The objects implemented in the POA can outlive the process in which they are first created.

Persistent objects have a POA associated with them which is the POA that created them. When the ORB receives a request on a persistent object, it first searches for the matching POA, based on the names of the POA and all of its ancestors.

### See Also

`PortableServer::LifespanPolicy`

### PortableServer::REQUEST\_PROCESSING\_POLICY\_ID Constant

```
//IDL
const CORBA::PolicyType REQUEST_PROCESSING_POLICY_ID = 22;
```

Defines an ID for the policy `RequestProcessingPolicy`.

### PortableServer::RequestProcessingPolicyValue Enumeration

```
// IDL
enum RequestProcessingPolicyValue {
    USE_ACTIVE_OBJECT_MAP_ONLY,
    USE_DEFAULT_SERVANT,
    USE_SERVANT_MANAGER
};
```

---

One of the following values can be supplied when creating a `RequestProcessingPolicy` policy.

<code>USE_ACTIVE_OBJECT_MAP_ONLY</code>	If the <code>ObjectId</code> is not found in the active object map, an <code>OBJECT_NOT_EXIST</code> exception is returned to the client. The <code>RETAIN</code> policy value is also required.
<code>USE_DEFAULT_SERVANT</code>	If the <code>ObjectId</code> is not found in the active object map or the <code>NON_RETAIN</code> policy value is present, and a default servant has been registered with the POA using <code>set_servant()</code> , the request is dispatched to the default servant. If no default servant has been registered, an <code>OBJ_ADAPTER</code> exception is returned to the client. The <code>MULTIPLE_ID</code> policy value is also required.
<code>USE_SERVANT_MANAGER</code>	If the <code>ObjectId</code> is not found in the active object map or the <code>NON_RETAIN</code> policy value is present, and a servant manager has been registered with the POA using <code>set_servant_manager()</code> , the servant manager is given the opportunity to locate a servant or raise an exception. If no servant manager has been registered, an <code>OBJECT_ADAPTER</code> exception is returned to the client.

## PortableServer::Servant Native Type

```
// IDL
native Servant;
```

The native `Servant` type has a language-specific mapping.

### PortableServer::SERVANT\_RETENTION\_POLICY\_ID Constant

```
//IDL
const CORBA::PolicyType SERVANT_RETENTION_POLICY_ID = 21;
```

Defines an ID for the policy `ServantRetentionPolicy`.

### PortableServer::ServantRetentionPolicyValue Enumeration

```
// IDL
enum ServantRetentionPolicyValue {
    RETAIN,
    NON_RETAIN
};
```

One of the following values can be supplied when creating a `ServantRetentionPolicy` policy.

RETAIN	The POA will retain active servants in its active object map.
NON_RETAIN	Servants are not retained by the POA. The NON_RETAIN policy requires either the USE_DEFAULT_SERVANT or USE_SERVANT_MANAGER policy values.

### PortableServer::THREAD\_POLICY\_ID Constant

```
//IDL
const CORBA::PolicyType THREAD_POLICY_ID = 16;
```

Defines an ID for the policy `ThreadPolicy`.

### PortableServer::ThreadPolicyValue Enumeration

```
//IDL
enum ThreadPolicyValue {
    ORB_CTRL_MODEL,
    SINGLE_THREAD_MODEL
};
```

---

One of the following values can be supplied when creating a `ThreadPolicy` policy.

- |                                  |   |
|----------------------------------|---|
| <code>ORB_CTRL_MODEL</code>      | The ORB is responsible for assigning requests for an ORB-controlled POA to threads. In a multi-threaded environment, concurrent requests may be delivered using multiple threads.   |
| <code>SINGLE_THREAD_MODEL</code> | Requests for a single-threaded POA are processed sequentially. In a multi-threaded environment, all up-calls made by this POA to implementation code (servants and servant managers) are made in a manner that is safe for code that is multi-thread-unaware. |

In some environments, using a value of `SINGLE_THREAD_MODEL` may mean that the POA will use only the main thread, in which case the application programmer is responsible to ensure that the main thread is given to the ORB, using `ORB::perform_work()` or `ORB::run()`. POAs using a value of `SINGLE_THREAD_MODEL` may need to cooperate to ensure that calls are safe even when a servant manager is shared by multiple single-threaded POAs. These models presume that the ORB and the application are using compatible threading primitives in a multi-threaded environment.

# PortableServer::AdapterActivator Interface

Adapter activators are associated with POAs. An adapter activator supplies a POA with the ability to create child POAs on demand, as a side-effect of receiving a request that names the child POA (or one of its children), or when `find_POA()` is called with an `activate` parameter value of `TRUE`. An application server that creates all its needed POAs at the beginning of execution does not need to use or provide an adapter activator; it is necessary only for the case in which POAs need to be created during request processing.

While a request from the POA to an adapter activator is in progress, all requests to objects managed by the new POA (or any descendant POAs) will be queued. This serialization allows the adapter activator to complete any initialization of the new POA before requests are delivered to that POA.

```
//IDL
interface AdapterActivator {
    boolean unknown\_adapter(
        in POA parent,
        in string name);
};
```

## AdapterActivator::unknown\_adapter()

```
//IDL
boolean
unknown_adapter(
    in POA parent,
    in string name
);

// Java
public boolean unknown_adapter(POA parent, java.lang.String name)
```

---

Recreates a POA `name` through the adapter activator of its parent POA. This method returns either true or false:

- True      The required POA is created; the ORB continues processing the request.
- False     The required POA was not created; the ORB returns an exception of `OBJECT_NOT_EXIST` to the client.

## Parameters

- `parent`      The parent POA.
- `name`         The new name for the POA.

This method is called on the parent POA's adapter activator when the ORB receives a request for an object reference whose POA does not exist. If the POA of the requested object has ancestor POAs that also no longer exist, the ORB calls this method on the adapter activator of each POA that must be recreated. The ORB iterates over the ancestral tree of the missing POA, starting with the most immediate existing ancestor—that is, the parent of the first missing POA. For each missing child POA (specified in parameter `name`), the ORB invokes this method on its parent's adapter activator until `name` resolves to the POA that contains the requested object reference.

For example, the ORB might seek an object reference in POA `x`, where `x` is descended from POA `b`, which in turn is a child of the root POA. If `b` and `x` no longer exist, the ORB must restore both POAs in order to recreate the context of the target object reference. By evaluating the object reference, it determines which POAs it needs to restore and calls `unknown_adapter()` on each one's parent:

1. Calls `unknown_adapter()` on the adapter activator of the root POA to recreate POA `b`.
2. If the first call to `unknown_adapter()` returns `TRUE`, the ORB calls `unknown_adapter()` on POA `b`'s adapter activator in order to recreate POA `x`.

Until this method returns, all requests to objects managed by the POAs that it creates are queued. If `unknown_adapter()` returns `FALSE`, ART replies to all queued requests with `OBJECT_NOT_EXIST`.



---

**Note:** `POA::find_POA()` calls this method if the POA to be found does not exist and its `activate_it` parameter is set to `TRUE`. If `unknown_adapter()` creates the POA and returns `TRUE`, `find_POA()` returns the required POA.

---

## Exceptions

<code>OBJECT_NOT_EXIST</code>	Raised by the ORB to the client if the parent of a POA that needs to be recreated does not have an adapter activator.
<code>OBJ_ADAPTER</code>	Raised to the client if the adapter activator raises a system exception.

## See Also

`PortableServer::POA::find_POA()`



# PortableServer::Current Interface

The `Current` interface, derived from `CORBA::Current`, provides method implementations with access to the identity of the object on which the method was invoked. The `Current` interface supports servants that implement multiple objects, but can be used within the context of POA-dispatched method invocations on any servant.

You obtain an instance of `Current` by calling `CORBA::ORB::resolve_initial_references("POACurrent")`. Thereafter, it can be used within the context of a method dispatched by the POA to obtain the POA and `ObjectId` that identify the object on which that operation was invoked.

```
//IDL
interface Current : CORBA::Current {
    exception NoContext {};
    POA get_POA() raises (NoContext);
    ObjectId get_object_id() raises (NoContext);
};
```

## Current::get\_object\_id()

```
//IDL
ObjectId get_object_id()
    raises(NoContext);

// Java
public byte[] get_object_id() throws NoContext
```

When called within the context of a request, this method returns the `ObjectId` of the target `CORBA` object.

Use this method to differentiate among different objects that map to the same servant, in a POA that has a `MULTIPLE_ID` policy value.

## Exceptions

<code>NoContext</code>	<code>get_object_id()</code> is called outside the context of a POA-dispatched operation.
------------------------	---

---

## Current::get\_POA()

```
//IDL
POA get_POA()
    raises(NoContext);
// Java
public POA get_POA() throws NoContext
```

When called within the context of a request, this method returns a reference to the POA that implements the target CORBA object.

### Exceptions

NoContext      `get_POA()` is called outside this context.

## Current::NoContext Exception

```
// IDL
exception NoContext {};
```

Indicates a `Current` method was called outside the context of POA-dispatched method invocations on a servant.

# PortableServer:: DynamicImplementation Class

DSI servants inherit from the standard `DynamicImplementation` class. This class inherits from the `Servant` class. The Dynamic Skeleton Interface (DSI) is implemented through servants that are members of classes that inherit from dynamic skeleton classes.

## DynamicImplementation::invoke()

```
// Java  
invoke(ServerRequest request)
```

The `invoke()` method receives requests issued to any CORBA object incarnated by the DSI servant and performs the processing necessary to execute the request.

The `invoke()` method should only be invoked by the POA in the context of serving a CORBA request. Invoking this method in other circumstances may lead to unpredictable results.



# PortableServer::IdAssignmentPolicy Interface

You obtain an `IdAssignmentPolicy` object by using `POA::create_id_assignment_policy()` and passing the policy to `POA::create_POA()` to specify whether `ObjectId` values in the created POA are generated by the application or by the ORB. This is a policy class derived from [CORBA::Policy](#).

If no `IdAssignmentPolicy` value is specified at POA creation, the default value is `SYSTEM_ID`.

```
// IDL
interface IdAssignmentPolicy : CORBA::Policy {
    readonly attribute IdAssignmentPolicyValue value;
};
```

## **IdAssignmentPolicy::value()**

```
// Java
public IdAssignmentPolicyValue value()
```

Returns the value of this POA policy.





# PortableServer::IdUniquenessPolicy Interface

You obtain an `IdUniquenessPolicy` object by using `POA::create_id_uniqueness_policy()` and passing the policy to `POA::create_POA()` to specify whether the servants activated in the created POA must have unique object identities. This is a policy class derived from [CORBA::Policy](#).

If no `IdUniquenessPolicy` value is specified at POA creation, the default value is `UNIQUE_ID`.

```
// IDL
interface IdUniquenessPolicy : CORBA::Policy {
    readonly attribute IdUniquenessPolicyValue value;
};
```

## **IdUniquenessPolicy::value()**

```
// Java
public IdUniquenessPolicyValue value()
```

Returns the value of this POA policy.



# PortableServer:: ImplicitActivationPolicy Interface

You obtain an `ImplicitActivationPolicy` object by using `POA::create_implicit_activation_policy()` and passing the policy to `POA::create_POA()` to specify whether implicit activation of servants is supported in the created POA. This is a policy class derived from [CORBA::Policy](#).

If no `ImplicitActivationPolicy` value is specified at POA creation, the default value is `NO_IMPLICIT_ACTIVATION`.

```
// IDL
interface ImplicitActivationPolicy : CORBA::Policy {
    readonly attribute ImplicitActivationPolicyValue value;
};
```

## **ImplicitActivationPolicy::value()**

```
// Java
public ImplicitActivationPolicyValue value()
```

Returns the value of this POA policy.



# PortableServer::LifespanPolicy Interface

You obtain a `LifespanPolicy` object by using `POA::create_lifespan_policy()` and passing the policy to `POA::create_POA()` to specify the lifespan of the objects implemented in the created POA. This is a policy class derived from [CORBA::Policy](#).

If no `LifespanPolicy` object is passed to `create_POA()`, the lifespan policy value defaults to `TRANSIENT`.

```
// IDL
interface LifespanPolicy : CORBA::Policy {
    readonly attribute LifespanPolicyValue value;
};
```

## **LifespanPolicy::value()**

```
// Java
public LifespanPolicyValue value()
```

Returns the value of this POA policy.



# PortableServer::POA Interface

A POA object manages the implementation of a collection of objects. The POA supports a name space for the objects, which are each identified by an `ObjectId`. A PPOAO also provides a name space for POAs. A POA is created as a child of an existing POA, which forms a hierarchy starting with the root POA.

```
//IDL
interface POA {
    exception AdapterAlreadyExists {};
    exception AdapterInactive {};
    exception AdapterNonExistent {};
    exception InvalidPolicy { unsigned short index; };
    exception NoServant {};
    exception ObjectAlreadyActive {};
    exception ObjectNotActive {};
    exception ServantAlreadyActive {};
    exception ServantNotActive {};
    exception WrongAdapter {};
    exception WrongPolicy {};

    //-----
    // POA creation and destruction
    //-----
    POA create\_POA(
        in string adapter_name,
        in POAManager a_POAManager,
        in CORBA::PolicyList policies
    )
        raises (AdapterAlreadyExists, InvalidPolicy);

    POA find\_POA(
        in string adapter_name,
        in boolean activate_it
    )
        raises (AdapterNonExistent);

    void destroy(
```

---

```

        in boolean etherealize_objects,
        in boolean wait_for_completion
    );

    // *****
    // Factories for Policy objects
    // *****
    ThreadPolicy create\_thread\_policy(
        in ThreadPolicyValue value
    );

    LifespanPolicy create\_lifespan\_policy(
        in LifespanPolicyValue value
    );

    IdUniquenessPolicy create\_id\_uniqueness\_policy(
        in IdUniquenessPolicyValue value
    );

    IdAssignmentPolicy create\_id\_assignment\_policy(
        in IdAssignmentPolicyValue value
    );

    ImplicitActivationPolicy create\_implicit\_activation\_policy(
        in ImplicitActivationPolicyValue value
    );

    ServantRetentionPolicy create\_servant\_retention\_policy(
        in ServantRetentionPolicyValue value
    );

    RequestProcessingPolicy create\_request\_processing\_policy(
        in RequestProcessingPolicyValue value
    );

    //-----
    // POA attributes
    //-----
    readonly attribute string the\_name;
    readonly attribute POA the\_parent;
    readonly attribute POAManager the\_POAManager;
    attribute AdapterActivator the\_activator;

```



---

```

//-----
// Servant Manager registration
//-----
ServantManager get\_servant\_manager()
    raises (WrongPolicy);

void set\_servant\_manager(in ServantManager imgr)
    raises (WrongPolicy);

//-----
// operations for the USE_DEFAULT_SERVANT policy
//-----
Servant get\_servant()
    raises (NoServant, WrongPolicy);

void set\_servant(in Servant servant)
    raises (WrongPolicy);

// *****
// object activation and deactivation
// *****
ObjectId activate\_object(in Servant servant)
    raises (ServantAlreadyActive, WrongPolicy);

void activate\_object\_with\_id(
    in ObjectId id,
    in Servant servant
)
    raises(
        ServantAlreadyActive,
        ObjectAlreadyActive,
        WrongPolicy
    );

void deactivate\_object(in ObjectId oid)
    raises (ObjectNotActive, WrongPolicy);

// *****
// reference creation operations
// *****
Object create\_reference(in CORBA::RepositoryId intf)
    raises (WrongPolicy);

```

---

```

Object create\_reference\_with\_id(
    in ObjectId oid,
    in CORBA::RepositoryId intf
)

//-----
// Identity mapping operations
//-----
ObjectId servant\_to\_id(in Servant servant)
    raises (ServantNotActive, WrongPolicy);

Object servant\_to\_reference(in Servant servant)
    raises (ServantNotActive, WrongPolicy);

Servant reference\_to\_servant(in Object reference)
    raises (ObjectNotActive, WrongAdapter, WrongPolicy);

ObjectId reference\_to\_id(in Object reference)
    raises (WrongAdapter, WrongPolicy);

Servant id\_to\_servant(in ObjectId oid)
    raises (ObjectNotActive, WrongPolicy);

Object id\_to\_reference(in ObjectId oid)
    raises (ObjectNotActive, WrongPolicy);
};

```

The exceptions defined for the POA class consists of the following:

```

AdapterAlreadyExists
AdapterInactive
AdapterNonExistent
InvalidPolicy
NoServant
ObjectAlreadyActive
ObjectNotActive
ServantAlreadyActive
ServantNotActive
WrongAdapter
WrongPolicy

```

The POA methods are described as follows:

---

## POA::activate\_object()

```
//IDL
ObjectId activate_object(in Servant servant)
    raises(ServantAlreadyActive, WrongPolicy);

// Java
public byte[] activate_object(Servant p_servant)
    throws ServantAlreadyActive, WrongPolicy
```

Returns a system-generated object ID and associates it with `servant` in the POA's active object map. This method can only be issued in a POA that has policies of `SYSTEM_ID` and `RETAIN`; otherwise, it raises an exception of `WrongPolicy`.

If the specified `servant` is already in the active object map and the POA has the `UNIQUE_ID` policy, the `ServantAlreadyActive` exception is raised.

### Exceptions

`ServantAlreadyActive`, `WrongPolicy`

### See Also

`PortableServer::POA::deactivate_object()`

## POA::activate\_object\_with\_id()

```
//IDL
void activate_object_with_id(
    in ObjectId oid,
    in Servant servant
)
    raises (
        ObjectAlreadyActive,
        ServantAlreadyActive,
        WrongPolicy);

// Java
public void activate_object_with_id(byte[] id, Servant p_servant)
    throws ServantAlreadyActive, ObjectAlreadyActive, WrongPolicy
```

Associates object `oid` with `servant` in the POA's active object map. This method can only be issued in a POA that has the `RETAIN` policy.

If you call `activate_object_with_id()` on a POA that has a policy of `SYSTEM_ID`, the object ID must be generated by that POA. To get the object ID of a system-generated object reference, call `reference_to_id()`.

---

## Exceptions

- ObjectAlreadyActive** Object `oid` is already active in this POA—that is, it is associated with a servant in the active object map.
- ServantAlreadyActive** The POA has the `UNIQUE_ID` policy and the servant is already associated with another object.
- WrongPolicy** The POA has the `NON_RETAIN` policy.

## POA::create\_id\_assignment\_policy()

```
//IDL
IdAssignmentPolicy create_id_assignment_policy(
    in IdAssignmentPolicyValue value
);
```

```
// Java
public IdAssignmentPolicy
    create_id_assignment_policy(IdAssignmentPolicyValue value)
```

Creates an object of the `IdAssignmentPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA. The ID assignment policy determines whether object IDs are generated by the POA or the application.

## Parameters

Specify the POA's ID assignment policy by supplying one of these values for the `value` parameter:

**SYSTEM\_ID:** (default) Only the POA can assign IDs to its objects. If the POA's lifespan policy is set to `PERSISTENT`, object IDs are unique across all instantiations of the same POA.

**USER\_ID:** Only the application can assign object IDs to objects in this POA. The application must ensure that all user-assigned IDs are unique across all instantiations of the same POA.

Typically, a POA with a `SYSTEM_ID` policy manages objects that are active for only a short period of time, and so do not need to outlive their server process. In this case, the POA also has an object lifespan policy of `TRANSIENT`.

---

`USER_ID` is usually assigned to a POA that has an object lifespan policy of `PERSISTENT`—that is, it generates object references whose validity can span multiple instantiations of a POA or server process, so the application requires explicit control over object IDs.

**See Also** `PortableServer::POA::create_poa()`

### **POA::create\_id\_uniqueness\_policy()**

```
//IDL
IdUniquenessPolicy create_id_uniqueness_policy(
    in IdUniquenessPolicyValue value
);

// Java
public IdUniquenessPolicy
    create_id_uniqueness_policy(IdUniquenessPolicyValue value)
```

Creates an object of the `IdUniquenessPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA. The ID uniqueness policy determines whether a servant can be associated with multiple objects in this POA.

**Parameters** Specify the POA's ID uniqueness policy by supplying one of these values for the `value` parameter:

**UNIQUE\_ID:** (default) Each servant in the POA can be associated with only one object ID.

**MULTIPLE\_ID:** Any servant in the POA can be associated with multiple object IDs.

---

**Note:** If the same servant is used by different POAs, that servant conforms to the uniqueness policy of each POA. Thus, it is possible for the same servant to be associated with multiple objects in one POA, and be restricted to one object in another.

---

**See Also** `PortableServer::POA::create_poa()`

---

## POA::create\_implicit\_activation\_policy()

```
//IDL
ImplicitActivationPolicy create_implicit_activation_policy(
    in ImplicitActivationPolicyValue value );

// Java
public ImplicitActivationPolicy create_implicit_activation_policy(
    ImplicitActivationPolicyValue value)
```

Creates an object of the `ImplicitActivationPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA. The activation policy determines whether the POA supports implicit activation of servants.

### Parameters

Specify the POA's activation policy by supplying one of these values for the value parameter:

**NO\_IMPLICIT\_ACTIVATION:** (default) The POA only supports explicit activation of servants.

**IMPLICIT\_ACTIVATION:** The POA supports implicit activation of servants. This policy requires that the POA's object ID assignment policy be set to `SYSTEM_ID`, and its servant retention policy be set to `RETAIN`.

### See Also

```
PortableServer::POA::create_poa()
PortableServer::POA::create_id_assignment_policy
PortableServer::POA::create_servant_retention_policy
```

## POA::create\_lifespan\_policy()

```
//IDL
LifespanPolicy create_lifespan_policy(
    in LifespanPolicyValue value
);

// Java
public LifespanPolicy create_lifespan_policy(
    LifespanPolicyValue value)
```

Creates an object of the `LifespanPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA. The lifespan policy determines whether object references outlive the process in which they were created.

---

**Parameters**

Specify a POA's lifespan policy by supplying one of these values for the `value` parameter:

**TRANSIENT:** (default) Object references do not outlive the POA in which they are created. After a transient object's POA is deactivated, attempts to reference this object yield the exception `CORBA:OBJECT_NOT_EXIST`

**PERSISTENT** Object references can outlive the POA in which they are created.

When a POA creates an object reference, it encapsulates it within an IOR. If the POA has a `TRANSIENT` policy, the IOR contains the server process's current location—its host address and port. Consequently, that object reference is valid only as long as the server process remains alive. If the server process dies, the object reference becomes invalid.

If the POA has a `PERSISTENT` policy, the IOR contains the address of the location domain's implementation repository, which maps all servers and their POAs to their current locations. Given a request for a persistent object, the Orbix daemon uses the object's "virtual" address first, and looks up the server process's actual location via the implementation repository.

A POA typically correlates its lifespan and ID assignment policies. `TRANSIENT` and `SYSTEM_ID` are the default settings for a new POA, out of recognition that system-assigned ID's are generally sufficient for transient object references. `PERSISTENT` and `USER_ID` policies are usually set together, inasmuch as an application typically requires explicit control over the object IDs of its persistent object references.

**See Also**

```
PortableServer::POA::create_poa()  
PortableServer::AdapterActivator::unknown_adapter()
```

**POA::create\_POA()**

```
//IDL  
POA create_POA(  
    in string adapter_name,  
    in POAManager a_POAManager,  
    in CORBA::PolicyList policies  
)  
    raises(AdapterAlreadyExists, InvalidPolicy);
```

---

```
// Java
public POA create_POA(
    java.lang.String adapter_name,
    POAManager a_POAManager,
    Policy[] policies)
    throws AdapterAlreadyExists, InvalidPolicy
```

Creates a portable object adapter (POA). An application calls this method on the parent of the new POA. The name of the new POA `adapter_name` must be unique among the names of all existing sibling POAs.

You control a POA's behavior through the policy objects that are created and attached to it through the `policies` parameter. A new POA has the following policy defaults

**Table 24:** *Policy Defaults for POAs*

Policy	Default Setting
<code>IdAssignmentPolicy</code>	<code>SYSTEM_ID</code>
<code>IdUniquenessPolicy</code>	<code>UNIQUE_ID</code>
<code>ImplicitActivationPolicy</code>	<code>NO_IMPLICIT_ACTIVATION</code>
<code>LifespanPolicy</code>	<code>TRANSIENT</code>
<code>RequestProcessingPolicy</code>	<code>USE_ACTIVE_OBJECT_MAP_ONLY</code>
<code>ServantRetentionPolicy</code>	<code>RETAIN</code>
<code>ThreadPolicy</code>	<code>ORB_CTRL_MODEL</code>

Policy objects are copied to the POA before this operation returns, so the application can destroy them while the POA is in use.

You can register either an existing POA manager or a new one with the POA through the `a_POAManager` parameter. If `a_POAManager` is null, a new `POAManager` object is registered with the POA. To obtain the `POAManager` object of the current POA, call `PortableServer::the_POAManager()`.

When you create a POA, it is in the state of its POA manager—typically, either active or holding. If you create a new POA manager with the POA, it is initially in a holding state. To process requests, it must be put into an active state through `PortableServer::POAManager::activate()`.



---

If you register an existing manager with the new POA and the manager is in an active state, the new POA might receive requests for objects before it is prepared to process them—that is, before its adapter activator, servant manager, or default servant is initialized. You can avoid this problem in one of these ways:

- Create the POA indirectly through its parent's adapter activator. For example, call `find_POA()` on the new POA's parent, supplying parameters `adapter_name` and `activate_it` arguments of the new (non-existent) POA and `TRUE`. Orbix queues all incoming requests on the new POA until the adapter activator returns on successful initialization of the POA.
- Before creating the POA, set its manager to a holding state through `PortableServer::POAManager::hold_requests()`.

## Exceptions

**AdapterAlreadyExists** The parent POA already has a child POA with the specified name.

**InvalidPolicy** Raised for one of these reasons:

- A policy object is not valid for the ORB implementation.
- Conflicting policy objects are specified—for example, `NON_RETAIN` and `USE_ACTIVE_OBJECT_MAP_ONLY`.

The exception index number specifies the first offending policy object through the corresponding index in the `policies` parameter.

## **POA::create\_reference()**

```
//IDL
Object create_reference(
    in CORBA::RepositoryId intf
)

// Java
public Object create_reference(java.lang.String intf)
```

Creates a CORBA object and returns an object reference. The object reference encapsulates a POA-generated object ID value and the specified interface

---

repository ID. This reference can be passed to clients so it can make requests on the corresponding object.

This operation requires the `SYSTEM_ID` policy. To obtain the generated object ID value call `POA::reference_to_id()` with the created reference. The returned object ID can then be used to associate the servant with an object (and thereby activate the object) by calling `activate_object_with_id()`.

**See Also**

`PortableServer::POA::create_reference_with_id`

### **POA::create\_reference\_with\_id()**

```
//IDL
Object create_reference_with_id(
    in ObjectId oid,
    in CORBA::RepositoryId intf
)

// Java
public Object create_reference_with_id(
    byte[] oid, java.lang.String intf)
```

Returns an object reference that encapsulates the specified object and interface repository identifiers. The resulting reference can be returned to clients to initiate requests on that object.

**See Also**

`PortableServer::POA::create_reference`

### **POA::create\_request\_processing\_policy()**

```
//IDL
RequestProcessingPolicy create_request_processing_policy(
    in RequestProcessingPolicyValue value
);

// Java
public RequestProcessingPolicy create_request_processing_policy(
    RequestProcessingPolicyValue value)
```

Creates an object of the `RequestProcessingPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA. This policy determines how the POA finds servants to implement requests.

Specify the POA's request processing policy by supplying one of these values:

- 
- `USE_ACTIVE_OBJECT_MAP_ONLY` (default): Assumes that all object IDs are mapped to a servant in the active object map. The active object map maintains an object-servant mapping until the object is explicitly deactivated through `deactivate_object()`.

This policy is typically used for a POA that processes requests for a small number of objects. If the object ID is not found in the active object map, an `OBJECT_NOT_EXIST` exception is raised to the client. This policy requires that the POA have a servant retention policy of `RETAIN`.

- `USE_DEFAULT_SERVANT`: Dispatch requests to the default servant when the POA cannot find a servant for the requested object. This can occur because the object's ID is not in the active object map, or the POA's servant retention policy is set to `NON_RETAIN`. This policy is typically used for a POA that needs to process many objects that are instantiated from the same class, and thus can be implemented by the same servant.

This policy requires that the application register a default servant with the POA via `set_servant()`; otherwise, an `OBJ_ADAPTER` exception is raised to the client. It also requires the POA's ID uniqueness policy be set to `MULTIPLE_ID`, so multiple objects can use the default servant.

- `USE_SERVANT_MANAGER`: The POA's servant manager finds a servant for the requested object when the object's ID is not in the active object map, or when the POA's servant retention policy is set to `NON_RETAIN`. If the servant manager fails to locate a servant, it raises an exception. This policy requires that the application register a servant manager with the POA via `set_servant_manager()`; otherwise, an `OBJ_ADAPTER` exception is returned to the client.

An application can implement either a servant activator or servant locator as a POA's servant manager, according to the POA's servant retention policy:

- A POA with a policy of `RETAIN` can register a servant activator. The servant activator incarnates servants for inactive objects on request; these objects remain active until the servant activator etherealizes them.
- A POA with a policy of `NON_RETAIN` can register a servant locator. The servant locator incarnates a servant for an inactive object each time the object is requested; the servant locator must etherealize the object and delete the servant from memory after the request returns.

A POA with a of `USE_SERVANT_MANAGER` policy allows the application to manage object activation directly.

---

**See Also**

```
PortableServer::POA::create_poa()  
PortableServer::POA::create_servant_retention_policy  
PortableServer::POA::create_id_uniqueness_policy
```

**POA::create\_servant\_retention\_policy()**

```
//IDL  
ServantRetentionPolicy create_servant_retention_policy(  
    in ServantRetentionPolicyValue value  
);  
  
// Java  
public ServantRetentionPolicy create_servant_retention_policy(  
    ServantRetentionPolicyValue value)
```

Creates an object of the `ServantRetentionPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA. This policy determines whether the POA has an active object map to maintain servant-object associations.

**Parameters**

Specify the servant retention policy by supplying one of these arguments for the `value` parameter:

**RETAIN:** (default) The POA retains active servants in its active object map. If combined with a policy of `USE_SERVANT_MANAGER`, the POA uses a servant activator as its servant manager.

**NON\_RETAIN:** The POA has no active object map. For each request, the POA relies on the servant manager or default servant to map between an object and its servant; all mapping information is destroyed when request processing returns. Therefore, a `NON_RETAIN` policy also requires that the POA have a request processing policy of `USE_DEFAULT_SERVANT` or `USE_SERVANT_MANAGER`.

**See Also**

```
PortableServer::POA::create_poa()  
PortableServer::POA::create_request_processing_policy()
```

**POA::create\_thread\_policy()**

```
//IDL  
ThreadPolicy create_thread_policy()
```

---

```
        in ThreadPolicyValue value
    );

    // Java
    public ThreadPolicy create_thread_policy(ThreadPolicyValue value)
```

Creates an object of the `ThreadPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA.

#### Parameters

Specify the POA's thread policy by supplying one of these values for the value parameter:

**ORB\_CTRL\_MODEL:** (default) The ORB is responsible for assigning requests for an ORB-controlled POA to threads. In a multi-threaded environment, concurrent requests can be delivered using multiple threads.

**SINGLE\_THREAD\_MODEL:** Requests for a single-threaded POA are processed sequentially. In a multi-threaded environment, all calls by a single-threaded POA to implementation code (servants and servant managers) are made in a manner that is safe for code that does not account for multi-threading.

Orbix uses the main thread for a single-threaded POA. In this case, make sure that the main thread is given to the ORB via `ORB::perform_work()` or `ORB::run()`. Multiple single-threaded POAs might need to cooperate to ensure that calls are safe when they share implementation code such as a servant manager.

Both threading policies assume that the ORB and the application are using compatible threading primitives in a multi-threaded environment. All uses of the POA within the server must conform to its threading policy.

#### See Also

```
PortableServer::POA::create_poa()
```

### POA::deactivate\_object()

```
//IDL
void deactivate_object(
    in ObjectId oid
)
    raises(ObjectNotActive, WrongPolicy);

// Java
public void deactivate_object(byte[] oid)
```

---

throws `ObjectNotActive`, `WrongPolicy`

Deactivates object `oid` by removing its servant association from the active object map. Call this method only for a POA with a `RETAIN` policy. If the POA has policies of `RETAIN` and `USE_SERVANT_MANAGER`, it calls the servant activator's `etherealize()` method. `deactivate_object()` returns immediately after its call to `etherealize()`.

## Exceptions

`ObjectNotActive` The specified object ID is not associated with a servant.  
e

`WrongPolicy` The POA has a `NON_RETAIN` policy.

## POA::destroy()

```
//IDL
void destroy(
    in boolean etherealize_objects
    in boolean wait_for_completion
);

// Java
public void destroy(boolean etherealize_objects,
    boolean wait_for_completion)
```

Destroys the target POA and all its descendant POAs. A POA thus destroyed can be recreated later on the same server process.

When a POA is destroyed, requests that already began execution on it or its descendents continue to completion. Requests that have not started execution are processed as if they were newly arrived—that is, the ORB tries to recreate the destroyed POA after all of its pending requests have finished processing.

`etherealize_objects` can be set to `TRUE` for a POA that has policies of `RETAIN` and `USE_SERVANT_MANAGER`. This parameter determines whether to call the servant activator's `etherealize()` method on each active object. Orbix perceives the POA to be destroyed, and therefore unavailable to requests, before any calls to `etherealize()` are made.

---

If `wait_for_completion` is set to `TRUE`, `destroy()` returns only after all requests in process and all calls to `etherealize()` return. If set to `FALSE`, `destroy()` returns after destroying the target POAs.

**See Also**

`PortableServer::POAManager::deactivate()`

## **POA::find\_POA()**

```
//IDL
POA find_POA(
    in string adapter_name,
    in boolean activate_it
)
    raises(AdapterNonExistent);

// Java
public POA find_POA(java.lang.String adapter_name,
    boolean activate_it)
    throws AdapterNonExistent
```

Returns a pointer to POA `adapter_name` if it is a child of the target POA. If the target POA has no child of the specified name and `activate_it` is set to `TRUE`, `find_POA()` invokes the target POA's adapter activator, if one exists. The adapter activator attempts to restore POA `adapter_name`; if successful, `find_POA()` returns the specified POA object.

**Exceptions**

`AdapterNonExistent` No POA is returned.

**See Also**

`PortableServer::AdapterActivator::unknown_adapter()`

## **POA::get\_servant()**

```
//IDL
Servant get_servant()
    raises(NoServant, WrongPolicy);

// Java
public Servant get_servant() throws NoServant, WrongPolicy
```

Returns the POA's default servant. This method can only be called on a POA that has the `USE_DEFAULT_SERVANT` policy.

---

## Exceptions

- NoServant      No default servant is associated with the POA.
- WrongPolicy    The POA should have the USE\_DEFAULT\_SERVANT policy.

### POA::get\_servant\_manager()

```
//IDL
ServantManager get_servant_manager()
    raises(WrongPolicy);

// Java
public ServantManager get_servant_manager() throws WrongPolicy
```

Returns the POA's servant manager. If no servant manager is associated with the POA, the method returns a null reference.

## See Also

PortableServer::AdapterActivator::set\_servant\_manager()

### POA::id\_to\_reference()

```
//IDL
Object id_to_reference(
    in ObjectId oid
)
    raises(ObjectNotActive, WrongPolicy);

// Java
public Object id_to_reference(byte[] oid)
    throws ObjectNotActive, WrongPolicy
```

Returns an object reference for active object `oid`. The object reference encapsulates the information used to direct requests to this object.

## Exceptions

- WrongPolicy    The POA has a policy of NON\_RETAIN policy.
- ObjectNotActive    The active object map does not contain the specified object ID.



---

## POA::id\_to\_servant()

```
//IDL
Servant id_to_servant(
    in ObjectId oid
)
    raises(ObjectNotActive, WrongPolicy);

// Java
public Servant id_to_servant(byte[] oid)
    throws ObjectNotActive, WrongPolicy
```

Returns the servant that is associated with object ID `oid` in the active object map. This method call is valid only for a POA with a `RETAIN` policy.

### Exceptions

`ObjectNotActive` The POA's active object map does not contain the specified object ID.

`WrongPolicy` The POA has a policy of `NON_RETAIN`.

## POA::reference\_to\_id()

```
//IDL
ObjectId reference_to_id(
    in Object reference
)
    raises WrongAdapter, WrongPolicy);

// Java
public byte[] reference_to_id(Object reference) throws
    WrongAdapter, WrongPolicy
```

Returns the object ID that is encapsulated by the specified object reference, where `reference` can specify an active or inactive object. Call this method only if the target POA created `reference`.

### Exceptions

`WrongAdapter` The object reference was not created by this POA.

`WrongPolicy` Reserved for future extensions.

---

## POA::reference\_to\_servant()

```
//IDL
Servant
reference_to_servant(
    in Object reference
)
    raises(ObjectNotActive, WrongAdapter, WrongPolicy);

// Java
public Servant reference_to_servant(Object reference)
    throws ObjectNotActive, WrongAdapter, WrongPolicy
```

Returns the servant that incarnates the *reference*-specified object if one of these conditions is true:

- The POA has a policy of `RETAIN` and the specified object is in the active object map.
- The POA has the `USE_DEFAULT_SERVANT` policy and a default servant is registered with the POA.

### Exceptions

<code>ObjectNotActive</code>	The POA policies are correct but no servant is associated with the specified object.
<code>WrongAdapter</code>	The object reference was not created by this POA.
<code>WrongPolicy</code>	The POA does not have a policy of either <code>RETAIN</code> or <code>USE_DEFAULT_SERVANT</code> .
<code>WrongAdapter</code>	The object reference was not created by this POA.

## POA::servant\_to\_id()

```
//IDL
ObjectId
servant_to_id(
    in Servant servant
)
    raises(ServantNotActive, WrongPolicy);

// Java
public byte[] servant_to_id(Servant p_servant) throws
    ServantNotActive, WrongPolicy
```

---

Returns an object ID for an object that is incarnated by *servant*.

Depending on the POA's policies, the method can take one of the following actions:

- Returns the ID of an already active object if the POA has the `UNIQUE_ID` policy and *servant* already incarnates an object.
- Associates *servant* with a POA-generated object ID and returns that ID if the POA has the `IMPLICIT_ACTIVATION` policy *and* one of these conditions is true:
  - ◆ The POA has the `MULTIPLE_ID` policy.
  - ◆ *servant* is not associated with any object.

## Exceptions

`WrongPolicy` The POA must have policies of `RETAIN` and either `UNIQUE_ID` or `IMPLICIT_ACTIVATION`; otherwise, it raises this exception. For example, if a POA has a policy of `RETAIN`, `NO_IMPLICIT_ACTIVATION`, and `MULTIPLE_ID`, `servant_to_id()` cannot tell which of the objects that this *servant* potentially incarnates it should return.

`ServantNotActive` *servant* is not associated with any object and the POA has a `NO_IMPLICIT_ACTIVATION` policy; therefore, no activation occurs.

## POA::servant\_to\_reference()

```
//IDL
Object servant_to_reference(
    in Servant servant
)
    raises (ServantNotActive, WrongPolicy);

// Java
public Object servant_to_reference(Servant p_servant)
    throws ServantNotActive, WrongPolicy
```

Returns an object reference for an object that is incarnated by *servant*.

Depending on the POA's policies, the method can take one of the following actions:

- 
- If the POA has the `UNIQUE_ID` policy and `servant` already incarnates an active object, the method returns an object reference that encapsulates the information used to activate that object.
  - If the POA has the `IMPLICIT_ACTIVATION` policy, and the POA has the `MULTIPLE_ID` policy or `servant` is not associated with any object, the servant is associated with a POA-generated object ID—thereby activating the object—and a corresponding object reference is returned.

## Exceptions

`WrongPolicy` The POA policy must have the `RETAIN` and either the `UNIQUE_ID` or `IMPLICIT_ACTIVATION` policies; otherwise, the exception is raised. For example, if a POA has a policy of `RETAIN`, `NO_IMPLICIT_ACTIVATION`, and `MULTIPLE_ID`, `servant_to_reference()` cannot ascertain which of the many objects potentially incarnated by the specified servant it should specify in its return.

`ServantNotActive` `servant` is not associated with any object and the POA has a `NO_IMPLICIT_ACTIVATION` policy; therefore, no activation occurs.

## POA::set\_servant()

```
//IDL
void
set_servant(
    in Servant servant
)
    raises(WrongPolicy);

// Java
public void set_servant(Servant p_servant) throws WrongPolicy
```

Registers `servant` with the POA as the default servant. This servant is used in a POA that has the `USE_DEFAULT_SERVANT` policy; it services any requests for objects that are not registered in the active object map

## Exceptions

`WrongPolicy` The POA does not have the `USE_DEFAULT_SERVANT` policy.

---

## POA::set\_servant\_manager()

```
//IDL
void set_servant_manager(
    in ServantManager imgr
)
    raises(WrongPolicy);

// Java
public void set_servant_manager(ServantManager imgr)
    throws WrongPolicy
```

Sets the default servant manager for the target POA.

### Exceptions

`WrongPolicy` Raised if the POA does not have a policy of `USE_SERVANT_MANAGER`.

## POA::the\_name()

```
// Java
```

Returns the name of the target POA.

## POA::the\_parent()

```
// Java
public POA the_parent()
```

Returns a pointer to the target POA's parent.

## POA::the\_POAManager()

```
// Java
public POAManager the_POAManager()
```

Returns a pointer to the target POA's manager.

---

## **POA::the\_activator()**

```
// Java  
public AdapterActivator the_activator()  
  
public void the_activator(AdapterActivator _val)
```

Returns or sets a pointer to the target POA's adapter activator.

# PortableServer::POAManager Interface

A `POAManager` is associated with one or more `POA` objects. (Each `POA` object has an associated `POAManager` object.) A `POA` manager encapsulates the processing state of its `POAs`. Using operations on the `POA` manager, an application can cause requests for those `POAs` to be queued or discarded, and can cause the `POAs` to be deactivated.

`POA` managers are created and destroyed implicitly. Unless an explicit `POAManager` object is provided at `POA` creation time, a `POA` manager is created when a `POA` is created and is automatically associated with that `POA`. A `POAManager` object is implicitly destroyed when all of its associated `POAs` have been destroyed.

```
//IDL
interface POAManager {
    exception AdapterInactive{};
    enum State { HOLDING, ACTIVE, DISCARDING, INACTIVE };
    void activate\(\)
        raises(AdapterInactive);
    void hold\_requests(
        in boolean wait_for_completion)
        raises(AdapterInactive);
    void discard\_requests(
        in boolean wait_for_completion)
        raises(AdapterInactive);
    void deactivate(
        in boolean etherealize_objects,
        in boolean wait_for_completion)
        raises(AdapterInactive);
    State get\_state\(\);
};
```

---

## POAManager::activate()

```
//IDL
void
activate()
    raises (AdapterInactive);

// Java
public void activate() throws AdapterInactive
```

Changes the state of the POA manager to active so it can process requests. When a POA manager is active, all associated POAs can receive requests.

---

**Note:** A POA manager's ability to process requests is dependent on resource limits. Orbix provides queues whose lengths are configurable, and raises a system exception of `TRANSIENT` when the queues are full.

---

When a POA manager is created, it is initially in a holding state. All requests sent to that POA are queued until you call `activate()` on a POA's manager. `activate()` can also reactivate a POA manager that has reverted to a holding state (due to a `hold_requests()` call) or is in a discarding state (due to a `discard_requests()` call).

If a new POA is associated with an existing active POA manager, it is unnecessary to call this method. However, it is generally, a good idea to put a POA manager in a holding state before creating a new POA with it.

### Exceptions

`AdapterInactive` This method is issued on an inactive POA manager.  
e

### See Also

```
PortableServer::POAManager::activate()
PortableServer::POAManager::deactivate()
PortableServer::POAManager::discard_requests()
PortableServer::POAManager::hold_requests()
```

## POAManager::AdapterInactive Exception

```
// IDL
exception AdapterInactive{};
```

Indicates that the `POAManager` is inactive and unable to process requests.



---

## POAManager::deactivate()

```
//IDL
void deactivate(
    in boolean etherealize_objects,
    in boolean wait_for_completion
);
    raises(AdapterInactive);

// Java
public void deactivate(boolean etherealize_objects,
    boolean wait_for_completion)
    throws AdapterInactive
```

Causes the POA manager to shut down. A POA manager deactivates before its associated POAs are destroyed. When it is in an inactive state, the POA manager allows all outstanding requests to complete processing, but refuses all incoming requests.

### Parameters

The method takes two boolean parameters:

<code>etherealize_objects</code>	Determines whether the target POAs etherealize their servants after all request processing is complete. This argument applies only to POAs that have a servant retention policy of <code>RETAIN</code> and request processing policy of <code>USE_SERVANT_MANAGER</code> .
<code>wait_for_completion</code>	Determines whether the method returns immediately or waits until the completion of all requests whose processing began before the call to <code>deactivate()</code> .

### Exceptions

<code>AdapterInactive</code>	The method is issued on a POA manager that is already inactive.
------------------------------	---

### See Also

`PortableServer::POA::destroy()`

## POAManager::discard\_requests()

```
//IDL
void discard_requests(
    in boolean wait_for_completion
)
```

---

```
        raises(AdapterInactive);  
  
    // Java  
    public void discard_requests(boolean wait_for_completion)  
        throws AdapterInactive
```

Causes the POA manager to discard all incoming requests. When a request is discarded, the server raises a `TRANSIENT` system exception to the client so it can reissue the request. This method can return immediately or wait until the return of all requests whose processing had already begun, according to the argument supplied for `wait_for_completion`.

### Parameters

`wait_for_completion` determines whether the method returns immediately or waits until the completion of all requests whose processing began before the call.

This method is typically called when an application detects that an object or the POA in general is overwhelmed by incoming requests. A POA manager should be in a discarding state only temporarily. On resolution of the problem that required this call, the application should restore the POA manager to its active state with `activate()`.

### Exceptions

`AdapterInactive` The method is issued on an inactive POA manager.  
e

### See Also

```
PortableServer::POAManager::activate()  
PortableServer::POAManager::discard_requests()  
PortableServer::POAManager::hold_requests()
```

## POAManager::get\_state()

```
//IDL  
State get_state();  
  
// Java  
public State get_state()
```

Returns the current state of the `POAManager`.

---

## POAManager::hold\_requests()

```
//IDL
void hold_requests(
    in boolean wait_for_completion
);
    raises(AdapterInactive);

// Java
public void hold_requests(boolean wait_for_completion) throws
    AdapterInactive
```

Causes all POAs associated with this manager to queue incoming requests.

The number of requests that can be queued is implementation-dependent. . Set `wait_for_completion` to determine whether this method returns immediately or waits until the return of all requests whose processing began before this call.

A POA manager is always created in a holding state.

### Exceptions

`AdapterInactive` The method is issued on an inactive POA manager.

### See Also

```
PortableServer::POAManager::activate()
PortableServer::POAManager::deactivate()
PortableServer::POAManager::discard_requests()
```

## POAManager::State Enumeration

```
//IDL
enum State { HOLDING, ACTIVE, DISCARDING, INACTIVE };
```

The possible state values consist of the following:

```
HOLDING
ACTIVE
DISCARDING
INACTIVE
```



# PortableServer:: RequestProcessingPolicy Interface

You obtain a `RequestProcessingPolicy` object by using `POA::create_request_processing_policy()` and passing the policy to `POA::create_POA()` to specify how requests are processed by the created POA. This is a policy class derived from [CORBA::Policy](#).

If no `RequestProcessingPolicy` value is specified at POA creation, the default value is `USE_ACTIVE_OBJECT_MAP_ONLY`.

You can define many possible combinations of behaviors with the policies `RequestProcessingPolicy` and `ServantRetentionPolicy`.

- `RETAIN` and `USE_ACTIVE_OBJECT_MAP_ONLY`  
This combination represents the situation where the POA does no automatic object activation (that is, the POA searches only the active object map). The server must activate all objects served by the POA explicitly, using either `activate_object()` or `activate_object_with_id()`.
- `RETAIN` and `USE_SERVANT_MANAGER`  
This combination represents a very common situation, where there is an active object map and a `ServantManager`. Because `RETAIN` is in effect, the application can call `activate_object()` or `activate_object_with_id()` to establish known servants in the active object map for use in later requests. If the POA doesn't find a servant in the active object map for a given object, it tries to determine the servant by means of invoking `incarnate()` in the `ServantManager` (specifically a `ServantActivator`) registered with the POA. If no `ServantManager` is available, the POA raises the `OBJECT_ADAPTER` system exception.
- `RETAIN` and `USE_DEFAULT_SERVANT`  
This combination represents the situation where there is a default servant defined for all requests involving unknown objects. Because `RETAIN` is in effect, the application can call `activate_object()` or `activate_object_with_id()` to establish known servants in the active

---

object map for use in later requests. The POA first tries to find a servant in the active object map for a given object. If it does not find such a servant, it uses the default servant. If no default servant is available, the POA raises the `OBJECT_ADAPTER` system exception.

- `NON-RETAIN` and `USE_SERVANT_MANAGER`  
This combination represents the situation where one servant is used per method call. The POA doesn't try to find a servant in the active object map because the active object map does not exist. In every request, it will call `preinvoke()` on the `ServantManager` (specifically a `ServantLocator`) registered with the POA. If no `ServantManager` is available, the POA will raise the `OBJECT_ADAPTER` system exception.
- `NON-RETAIN` and `USE_DEFAULT_SERVANT`  
This combination represents the situation where there is one single servant defined for all CORBA objects. The POA does not try to find a servant in the active object map because the active object map doesn't exist. In every request, the POA will invoke the appropriate operation on the default servant registered with the POA. If no default servant is available, the POA will raise the `OBJECT_ADAPTER` system exception.

```
// IDL
interface RequestProcessingPolicy : CORBA::Policy {
    readonly attribute RequestProcessingPolicyValue value;
};
```

### **RequestProcessingPolicy::value()**

```
// Java
public RequestProcessingPolicyValue value()
```

Returns the value of this POA policy.

# PortableServer::ServantActivator Interface

When a POA has the `RETAIN` policy value, it uses the `ServantActivator` type of servant manager.

```
//IDL
interface ServantActivator : ServantManager {
    Servant incarnate(
        in ObjectId oid,
        in POA adapter
    )
    raises (ForwardRequest);

    void etherealize(
        in ObjectId oid,
        in POA adapter,
        in Servant serv,
        in boolean cleanup_in_progress,
        in boolean remaining_activations
    );
};
```

## ServantActivator::etherealize()

```
//IDL
void etherealize(
    in ObjectId oid,
    in POA adapter,
    in Servant serv
    in boolean cleanup_in_progress
    in boolean remaining_activations
);

// Java
public void etherealize(byte[] oid,
    POA adapter,
```

---

```
Servant serv,  
boolean cleanup_in_progress,  
boolean remaining_activations)
```

Destroys a servant in a POA that has the `RETAIN` and `USE_SERVANT_MANAGER` policy values.

The `cleanup_in_progress` parameter indicates the context in which this method was called. If is set to true, `etherealize()` was called because of calls to either `PortableServer::POAManager::deactivate()` or `PortableServer::POA::destroy()` with its `etherealize_objects` parameter set to true. If `cleanup_in_progress` is false, this method was called for other reasons.

Because a servant can incarnate multiple objects, `etherealize()` checks the `remaining_activations` parameter to determine whether this servant incarnates any other objects within this POA; if `remaining_activations` is set to `FALSE` and the servant is not used by other POAs, the method can safely delete the servant from memory.

Before the POA calls on a servant manager's `etherealize()` method, it takes steps to ensure the safe destruction of servants in a multi-threaded environment:

- Removes the target object and its servant from the active object map. Thus, new requests for the target object cannot be invoked on the servant while it undergoes etherealization.
- Calls `etherealize()` on the servant only after all outstanding requests finish processing.

A servant can be etherealized by a servant activator other than the one that originally incarnated it.

### **ServantActivator::incarnate()**

```
//IDL  
Servant incarnate(  
    in ObjectId oid,  
    in POA adapter  
)  
    raises (ForwardRequest);
```



---

```
// Java
public Servant incarnate(byte[] oid, POA adapter)
    throws ForwardRequest
```

Called by the POA when it receives a request for object `oid`, where `oid` contains the ID of an inactive object. `incarnate()` returns an appropriate servant for the requested object; this servant is associated with `oid` in the POA's active object map, thereby activating the object. Subsequent requests for this object are mapped directly to the servant.

This method is only called by the POA on a servant activator, which the POA uses as its servant manager when it has policies of `USE_SERVANT_MANAGER` and `RETAIN`. When using a servant activator, the active object map retains a servant-object association until the servant is etherealized. A servant can only incarnate a given object once. If the POA has a policy of `UNIQUE_ID`, `incarnate()` can only return a servant that does not incarnate any object other than `oid` in that POA.

---

**Note:** If the same servant is used by different POAs, that servant conforms to the uniqueness policy of each POA. Thus, it is possible for the same servant to be associated with multiple objects in one POA, and be restricted to one object in another.

---

## Exceptions

`ForwardRequest` The client is instructed to send this request and subsequent requests for `oid` to the object specified in the exception's `forward_reference` member—in IIOP, through a `LOCATION_FORWARD` reply.

## See Also

`PortableServer::ServantActivator::etherealize()`  
`PortableServer::ServantLocator::preinvoke()`



# PortableServer::ServantBase

**(Title Variable)**

---

# PortableServer::ServantLocator Interface

When the POA has the `NON_RETAIN` policy value it uses servant locators as its servant managers. Because the POA knows that the servant returned by a `ServantLocator` will be used only for a single request, it can supply extra information to the servant manager's operations. Also, the servant manager's pair of operations may be able to cooperate to do something different than a `ServantActivator`.

```
//IDL
interface ServantLocator : ServantManager {
    native Cookie;
    Servant preinvoke(
        in ObjectId oid,
        in POA adapter,
        in CORBA::Identifier operation,
        out Cookie the_cookie
    ) raises (ForwardRequest);

    void postinvoke(
        in ObjectId oid,
        in POA adapter,
        in CORBA::Identifier operation,
        in Cookie the_cookie,
        in Servant the_servant
    );
};
```

## ServantLocator::Cookie Native Type

```
// IDL
native Cookie;
```

The `Cookie` native type is opaque to the POA. It can be set by the servant manager for use later by `postinvoke()`.

---

## ServantLocator::postinvoke()

```
//IDL
void postinvoke(
    in ObjectId oid,
    in POA adapter
    in CORBA::Identifier operation,
    in Cookie the_cookie,
    in Servant the_servant
);

// Java
public void postinvoke(byte[] oid,
    POA adapter,
    java.lang.String operation,
    java.lang.Object the_cookie,
    Servant the_servant)
```

Called on a POA's servant locator to delete a servant when processing of a request for object `oid` is complete.

Each `postinvoke()` call is paired to an earlier `preinvoke()` call. In order to explicitly map data between the two calls, you set the `preinvoke()` method's `the_cookie` parameter. This can be especially useful in a multi-threaded environment where it is important to ensure that a pair of `preinvoke()` and `postinvoke()` calls operate on the same servant. For example, each `preinvoke()` call can set its `the_cookie` parameter to data that identifies its servant; the `postinvoke()` code can then compare that data to its `the_servant` parameter.

The POA calls this method only on a servant locator, which the POA uses as its servant manager when it has policies of `USE_SERVANT_MANAGER` and `NON_RETAIN`.

### See Also

```
PortableServer::ServantLocator::preinvoke()
PortableServer::POA::create_reference_with_id()
```

## ServantLocator::preinvoke()

```
//IDL
Servant preinvoke(
    in ObjectId oid,
    in POA adapter,
```

---

```
        in CORBA::Identifier operation,
        out Cookie the_cookie
    )
    raises (ForwardRequest);

// Java
public Servant preinvoke(byte[] oid,
    POA adapter,
    java.lang.String operation,
    CookieHolder the_cookie)
    throws ForwardRequest
```

Returns an appropriate servant for the requested object. This method is called on a POA's servant locator when the POA receives a request for object `oid`, where `oid` contains the ID of an inactive object.

This method is only called by the POA on a servant locator, which the POA uses as its servant manager when it has policies of `USE_SERVANT_MANAGER` and `NON_RETAIN`.

The lack of an active object map can require the following behavior:

- After processing on the requested object is complete, the POA calls `postinvoke()` on the object and etherealizes its servant.
- Each request for an object is treated independently, irrespective of the status of earlier requests for that object. So, it is possible for a POA to accept multiple requests for the same object concurrently and for its servant locator to incarnate several servants for that object simultaneously.

Alternatively, the application can maintain its own object-servant map in order to allow a servant to process multiple requests for the same object, or to process requests for multiple objects. For example, a database server can use a servant locator to direct concurrent operations to the same servant; database transactions are opened and closed within the `preinvoke()` and `postinvoke()` operations.

Each `preinvoke()` call is paired to an later `postinvoke()` call. In order to explicitly map data between the two calls, set `preinvoke()`'s `the_cookie` parameter. This can be especially useful in a multi-threaded environment where it is important to ensure that a pair of `preinvoke()` and `postinvoke()` calls operate on the same servant. For example, each `preinvoke()` call can set its cookie parameter to data that identifies its servant; the `postinvoke()` code can then compare that data to its `the_servant` parameter.

---

## Exceptions

`ForwardRequest` The client is instructed to send this request and subsequent requests for `oid` to the object specified in the exception's `forward_reference` member—in IIOP, through a `LOCATION_FORWARD` reply.

## See Also

`PortableServer::ServantLocator::postinvoke()`



# PortableServer::ServantManager Interface

A servant manager supplies a POA with the ability to activate objects on demand when the POA receives a request targeted at an inactive object. A servant manager is registered with a POA as a callback object, to be invoked by the POA when necessary.

A servant manager is used in servers only for the case in which an object must be activated during request processing. An application server that activates all its needed objects at the beginning of execution does not need to use a servant manager.

The `ServantManager` interface is an empty base interface that is inherited by the interfaces `ServantActivator` and `ServantLocator`. These two types of servant managers have the following corresponding policy values:

**Table 25:** *Corresponding Policies for Servant Managers*

Servant Manager	POA Policy Value
<code>ServantActivator</code>	RETAIN
<code>ServantLocator</code>	NON_RETAIN

```
//IDL
interface ServantManager
{ };

PortableServer::ServantActivator
PortableServer::ServantLocator
```

## See Also



# PortableServer:: ServantRetentionPolicy Interface

You obtain a `ServantRetentionPolicy` object by using `POA::create_servant_retention_policy()` and passing the policy to `POA::create_POA()` to specify whether the created POA retains active servants in an active object map. This is a policy class derived from [CORBA::Policy](#).

If no `ServantRetentionPolicy` value is specified at POA creation, the default value is `RETAIN`.

## See Also

`PortableServer::RequestProcessingPolicy`

```
// IDL
interface ServantRetentionPolicy : CORBA::Policy {
    readonly attribute ServantRetentionPolicyValue value;
};
```

## ServantRetentionPolicy::value()

```
// Java
public ServantRetentionPolicyValue value()
```

Returns the value of this POA policy.



# PortableServer::ThreadPolicy Interface

You obtain a `ThreadPolicy` object by using `POA::create_thread_policy()` and passing the policy to `POA::create_POA()` to specify the threading model used with the created POA. This is a policy class derived from [CORBA::Policy](#).

```
// IDL
interface ThreadPolicy : CORBA::Policy {
    readonly attribute ThreadPolicyValue value;
};
```

## ThreadPolicy::value()

```
// Java
public ThreadPolicyValue value()
```

Returns the value of this POA policy.



# Security Overview

The standard `Security` module defines data types and constants that are used throughout the CORBA security specification. This section documents only the definitions relevant to Orbix SSL/TLS.

There is also a reference in [Javadoc format](#).

## Security::AssociationOptions Type

```
// IDL
typedef unsigned short AssociationOptions;
```

A data type that holds a set of association options in its bit fields.

### See Also

Security::[NoProtection](#)  
Security::[Integrity](#)  
Security::[Confidentiality](#)  
Security::[DetectReplay](#)  
Security::[DetectMisordering](#)  
Security::[EstablishTrustInTarget](#)  
Security::[EstablishTrustInClient](#)  
Security::[NoDelegation](#)  
Security::[SimpleDelegation](#)  
Security::[CompositeDelegation](#)

## Security::AttributeList Sequence

```
// IDL
typedef sequence <SecAttribute> AttributeList;
```

---

## Security::AuthenticationMethod Type

```
// IDL
typedef unsigned long AuthenticationMethod;
```

Constants of this type are used by the [SecurityLevel2::PrincipalAuthenticator::authenticate\(\)](#) operation to identify an authentication method. Orbix SSL/TLS defines a range of AuthenticationMethod constants in the [IT\\_TLS\\_API](#) module—for example, [IT\\_TLS\\_API::IT\\_TLS\\_AUTH\\_METH\\_PKSC12\\_FILE](#).

## Security::AuthenticationMethodList Sequence

```
// IDL
typedef sequence<AuthenticationMethod> AuthenticationMethodList;
```

A list of authentication methods.

## Security::AuthenticationStatus Enumeration

```
// IDL
enum AuthenticationStatus {
    SecAuthSuccess,
    SecAuthFailure,
    SecAuthContinue,
    SecAuthExpired
};
```

Used by the [SecurityLevel2::PrincipalAuthenticator::authenticate\(\)](#) operation to give the status of the returned credentials.

### Values

The status of a newly-generated [Credentials](#) object, [creds](#), is indicated as follows:

SecAuthSuccess	A valid <a href="#">Credentials</a> object is available in the <a href="#">creds</a> parameter.
SecAuthFailure	Authentication was in some way inconsistent or erroneous. Credentials have therefore not been created.



---

<code>SecAuthContinue</code>	The authentication procedure uses a challenge and response mechanism. The <code>creds</code> parameter references a partially initialized <code>Credentials</code> object and the <code>continuation_data</code> indicates details of the challenge. Not supported by Orbix SSL/TLS.
<code>SecAuthExpired</code>	The authentication data, <code>auth_data</code> , has expired. Credentials have therefore not been created.

### Security::CommunicationDirection Enumeration

```
// IDL
enum CommunicationDirection {
    SecDirectionBoth,
    SecDirectionRequest,
    SecDirectionReply
};
```

Indicates a particular communication direction along a secure association.

#### See Also

[SecurityLevel2::Credentials::get\\_security\\_feature\(\)](#)

### Security::CompositeDelegation Constant

```
// IDL
const AssociationOptions CompositeDelegation = 512;
```

Not supported in Orbix SSL/TLS.

### Security::Confidentiality Constant

```
// IDL
const AssociationOptions Confidentiality = 4;
```

Specifies that an object supports or requires confidentiality-protected invocations.

---

## Security::DetectMisordering Constant

```
// IDL
const AssociationOptions DetectMisordering = 16;
```

Specifies that an object supports or requires error detection on fragments of invocation messages. In Orbix SSL/TLS this option can be set only through configuration.

## Security::DetectReplay Constant

```
// IDL
const AssociationOptions DetectReplay = 8;
```

Specifies that an object supports or requires replay detection on invocation messages. In Orbix SSL/TLS this option can be set only through configuration.

## Security::EstablishTrust Structure

```
// IDL
struct EstablishTrust {
    boolean trust_in_client;
    boolean trust_in_target;
};
```

### Parameters

This structure is used to hold the data associated with the [SecurityLevel2::EstablishTrustPolicy](#).

The elements of the structure are, as follows:

<code>trust_in_client</code>	Specifies whether or not an invocation must select credentials and a mechanism that allow the client to be authenticated to the target. (Some mechanisms might not support client authentication).
<code>trust_in_target</code>	Specifies whether or not an invocation must establish trust in the target.

---

## Security::EstablishTrustInClient Constant

```
// IDL
const AssociationOptions EstablishTrustInClient = 64;
```

Specifies that a client supports or requires that the target authenticate its identity to the client.

**See Also** [SecurityLevel2::EstablishTrustPolicy](#)

## Security::EstablishTrustInTarget Constant

```
// IDL
const AssociationOptions EstablishTrustInTarget = 32;
```

Specifies that a target object requires the client to authenticate its privileges to the target.

**See Also** [SecurityLevel2::EstablishTrustPolicy](#)

## Security::Integrity Constant

```
// IDL
const AssociationOptions Integrity = 2;
```

Specifies that an object supports integrity-protected invocations.

## Security::InvocationCredentialsType Enumeration

```
// IDL
enum InvocationCredentialsType {
    SecOwnCredentials,
    SecReceivedCredentials,
    SecTargetCredentials
};
```

Identifies the underlying type of a [SecurityLevel2::Credentials](#) object, as follows:

SecOwnCredentials	The underlying type is <a href="#">SecurityLevel2::Credentials</a> .
-------------------	--

---

SecReceivedCredentials The underlying type is [SecurityLevel2::ReceivedCredentials](#).

SecTargetCredentials The underlying type is [SecurityLevel2::TargetCredentials](#).

## Security::MechanismType Type

```
// IDL
typedef string MechanismType;
```

Identifies a security mechanism.

**See Also**

[SecurityLevel2::MechanismPolicy](#)

## Security::MechanismTypeList Sequence

```
// IDL
typedef sequence<MechanismType> MechanismTypeList;
```

A list of security mechanisms.

**See Also**

[SecurityLevel2::MechanismPolicy](#)

## Security::NoDelegation Constant

```
// IDL
const AssociationOptions NoDelegation = 128;
```

Not supported in Orbix SSL/TLS.

## Security::NoProtection Constant

```
// IDL
const AssociationOptions NoProtection = 1;
```

When used with the target secure invocation policy, indicates that the target can accept insecure connections.

---

When used with the client secure invocation policy, indicates that the client can open insecure connections.

## Security::Opaque Type

```
// IDL
typedef sequence <octet> Opaque;
```

A general purpose type that is used to hold binary data.

## Security::QOP Enumeration

```
// IDL
enum QOP {
    SecQOPNoProtection,
    SecQOPIntegrity,
    SecQOPConfidentiality,
    SecQOPIntegrityAndConfidentiality
};
```

Identifies the range of security features that can be associated with an individual object reference (quality of protection).

SecQOPNoProtection	The Security:: <a href="#">NoProtection</a> association option.
SecQOPIntegrity	The Security:: <a href="#">Integrity</a> association option.
SecQOPConfidentiality	The Security:: <a href="#">Confidentiality</a> association option.
SecQOPIntegrityAndConfidentiality	Both the Security:: <a href="#">Integrity</a> and Security:: <a href="#">Confidentiality</a> association options.

## Security::SecApplicationAccess Constant

```
// IDL
const CORBA::PolicyType SecApplicationAccess = 3;
```

---

Not supported in Orbix SSL/TLS.

### **Security::SecAttribute Structure**

```
// IDL
struct SecAttribute {
    AttributeType attribute_type;
    OID defining_authority;
    Opaque value;
};
```

### **Security::SecClientInvocationAccess Constant**

```
// IDL
const CORBA::PolicyType SecClientInvocationAccess = 1;
```

Not supported in Orbix SSL/TLS.

### **Security::SecClientSecureInvocation Constant**

```
// IDL
const CORBA::PolicyType SecClientSecureInvocation = 8;
```

Defines one of the policy types for the `SecurityAdmin::SecureInvocationPolicy` interface. This policy can only be set through configuration.

### **Security::SecEstablishTrustPolicy Constant**

```
// IDL
const CORBA::PolicyType SecEstablishTrustPolicy = 39;
```

Defines the policy type for the [SecurityLevel2::EstablishTrustPolicy](#) interface.

---

## Security::SecInvocationCredentialsPolicy Constant

```
// IDL
const CORBA::PolicyType SecInvocationCredentialsPolicy = 13;
```

Defines the policy type for the [SecurityLevel2::InvocationCredentialsPolicy](#) interface.

## Security::SecMechanismsPolicy Constant

```
// IDL
const CORBA::PolicyType SecMechanismsPolicy = 12;
```

Defines the policy type for the [SecurityLevel2::MechanismsPolicy](#) interface.

### See Also

```
IT_TLS_API::TLS::create_mechanism_policy()
```

## Security::SecQOPPolicy Constant

```
// IDL
const CORBA::PolicyType SecQOPPolicy = 15;
```

Defines the policy type for the [SecurityLevel2::QOPPolicy](#) interface.

## Security::SecTargetInvocationAccess Constant

```
// IDL
const CORBA::PolicyType SecTargetInvocationAccess = 2;
```

Not supported in Orbix SSL/TLS.

## Security::SecTargetSecureInvocation Constant

```
// IDL
const CORBA::PolicyType SecTargetSecureInvocation = 9;
```

Defines one of the policy types for the [SecurityAdmin::SecureInvocationPolicy](#) interface. This policy can only be set through configuration.

---

## Security::SecurityFeature Enumeration

```
// IDL
enum SecurityFeature {
    SecNoDelegation,
    SecSimpleDelegation,
    SecCompositeDelegation,
    SecNoProtection,
    SecIntegrity,
    SecConfidentiality,
    SecIntegrityAndConfidentiality,
    SecDetectReplay,
    SecDetectMisordering,
    SecEstablishTrustInTarget,
    SecEstablishTrustInClient
};
```

Identifies the range of security features that can be associated with a [Credentials](#) object, including association options.

### Values

This enumeration can have the following values:

SecNoDelegation	The Security:: <a href="#">NoDelegation</a> association option.
SecSimpleDelegation	The Security:: <a href="#">SimpleDelegation</a> association option. Not supported in Orbix SSL/TLS.
SecCompositeDelegation	The Security:: <a href="#">CompositeDelegation</a> association option. Not supported in Orbix SSL/TLS.
SecNoProtection	The Security:: <a href="#">NoProtection</a> association option.
SecIntegrity	The Security:: <a href="#">Integrity</a> association option.
SecConfidentiality	The Security:: <a href="#">Confidentiality</a> association option.
SecIntegrityAndConfidentiality	Both the Security:: <a href="#">Integrity</a> and Security:: <a href="#">Confidentiality</a> association options.



---

SecDetectReplay	The Security:: <a href="#">DetectReplay</a> association option.
SecDetectMisordering	The Security:: <a href="#">DetectMisordering</a> association option.
SecEstablishTrustInTarget	The Security:: <a href="#">EstablishTrustInTarget</a> association option.
SecEstablishTrustInClient	The Security:: <a href="#">EstablishTrustInClient</a> association option.

**See Also** [SecurityLevel2::Credentials::get\\_security\\_feature\(\)](#)  
[Security::AssociationOptions](#)

### Security::SecurityName Type

```
// IDL
typedef string SecurityName;
```

A string that identifies a principal (for example, a login name).

Not used by Orbix SSL/TLS.

### Security::SimpleDelegation Constant

```
// IDL
const AssociationOptions SimpleDelegation = 256;
```

Not supported in Orbix SSL/TLS.



# SecurityLevel1 Overview

Because security level 1 is aimed at security-unaware applications, there is little IDL defined at this level—most of the security features are controlled by an administrator. Currently, there is one IDL interface defined at level 1:

- SecurityLevel1::[Current](#)



# SecurityLevel1::Current Interface

**IDL**

```
// IDL in module SecurityLevel1
local interface Current : CORBA::Current { // Locality Constrained
    // thread specific operations
    Security::AttributeList get_attributes (
        in Security::AttributeTypeList attributes
    );
};
```

**Description** The Current object enables you to access information about the execution context. In Orbix SSL/TLS, it enables a server object to access a client's credentials.

## Current::get\_attributes()

**IDL**

```
Security::AttributeList get_attributes (
    in Security::AttributeTypeList attributes
);
```

**Description** Not implemented in Orbix SSL/TLS.

You can use the `Credentials::get_attributes()` operation instead.

**See Also** `SecurityLevel2::Current::received_credentials`  
`SecurityLevel2::Credentials::get_attributes()`



# SecurityLevel2 Overview

At security level 2, IDL interfaces are defined to enable security-aware application to access security information and specify security policies. Orbix SSL/TLS implements the following IDL interfaces from the `SecurityLevel2` IDL module:

- [PrincipalAuthenticator](#) interface.
- [Credentials](#) interface.
- [ReceivedCredentials](#) interface.
- [TargetCredentials](#) interface.
- [QOPPolicy](#) interface.
- [MechanismPolicy](#) interface.
- [InvocationCredentialsPolicy](#) interface.
- [EstablishTrustPolicy](#) interface.
- [SecurityManager](#) interface.
- [Current](#) interface.

## SecurityLevel2::CredentialsList Sequence

```
// IDL
typedef sequence <Credentials> CredentialsList;
```

A sequence to hold a list of `Credentials` objects.





# SecurityLevel2::Credentials Interface

IDL

```
// IDL in module SecurityLevel2
interface Credentials {           // Locality Constrained
# pragma version Credentials 1.7
    Credentials copy();

    void destroy();

    readonly attribute Security::InvocationCredentialsType
        credentials_type;
    readonly attribute Security::AuthenticationStatus
        authentication_state;
    readonly attribute Security::MechanismType
        mechanism;

    attribute Security::AssociationOptions
        accepting_options_supported;
    attribute Security::AssociationOptions
        accepting_options_required;
    attribute Security::AssociationOptions
        invocation_options_supported;
    attribute Security::AssociationOptions
        invocation_options_required;

    boolean get_security_feature(
        in Security::CommunicationDirection direction,
        in Security::SecurityFeature feature
    );

    boolean set_attributes (
        in Security::AttributeList requested_attributes,
        out Security::AttributeList actual_attributes
    );

    Security::AttributeList get_attributes (
        in Security::AttributeTypeList attributes
    );
};
```

---

```
    boolean is_valid (out Security::UtcT expiry_time);

    boolean refresh(in any refresh_data);
};
```

**Description** The `Credentials` interface is used either as a base interface or as a concrete interface (most derived type is `Credentials`). An object of `Credentials` type can represent one of the following kinds of credential:

- *Own credentials*—when the most derived type of the `Credentials` object is `Credentials`.
- *Received credentials*—when the most derived type of the `Credentials` object is [ReceivedCredentials](#).
- *Target credentials*—when the most derived type of the `Credentials` object is [TargetCredentials](#).

A `Credentials` object holds the security attributes of a principal.

**See Also** `IT_TLS_API::TLSCredentials`  
`IT_TLS_API::TLSReceivedCredentials`  
`IT_TLS_API::TLSTargetCredentials`

### **Credentials::accepting\_options\_required Attribute**

**IDL** attribute [Security::AssociationOptions](#) `accepting_options_required;`

**Description** Not implemented in Orbix SSL/TLS.

### **Credentials::accepting\_options\_supported Attribute**

**IDL** attribute [Security::AssociationOptions](#)  
`accepting_options_supported;`

**Description** Not implemented in Orbix SSL/TLS.

### **Credentials::authentication\_state Attribute**

**IDL** readonly attribute [Security::AuthenticationStatus](#)  
`authentication_state;`

---

**Description** Specifies how a `Credentials` object is initialized (authentication state) at the time it is created by the [PrincipalAuthenticator](#) object.

**Values** The authentication state can have one of the following values:

`SecAuthSuccess` The `Credentials` object is fully initialized and valid.

`SecAuthExpired` The credentials initialization has expired and the credentials are invalid.

### **Credentials::copy()**

**IDL** `Credentials copy();`

**Description** Returns a reference to a deep copy of the target `Credentials` object.  
Not implemented in Orbix SSL/TLS.

### **Credentials::credentials\_type Attribute**

**IDL** `readonly attribute Security::InvocationCredentialsType  
credentials_type;`

**Description** Indicates whether the `Credentials` object represents an application's own credentials (of `Credentials` type), or received credentials (of [ReceivedCredentials](#) type), or target credentials (of [TargetCredentials](#) type).

**Values** This attribute can have one of the following values:

[Security::SecOwnCredentials](#) Indicates own credentials

[Security::SecReceivedCredentials](#) Indicates received credentials.

[Security::SecTargetCredentials](#) indicates target credentials

---

## Credentials::destroy()

- IDL** `void destroy();`
- Description** Destroys the `Credentials` object.  
Not implemented in Orbix SSL/TLS.

## Credentials::get\_attributes()

- IDL** `Security::AttributeList get_attributes(  
in AttributeTypeList attributes  
);`
- Description** Returns the security attributes from a `Credentials` object.
- Parameters** This operation takes the following parameter:
- |                         |  |
|-------------------------|--|
| <code>attributes</code> | The set of security attributes (attributes and identities) whose values are desired. If this list is empty, all attributes are returned. |
|-------------------------|--|

## Credentials::get\_security\_feature()

- IDL** `boolean get_security_feature(  
in Security::CommunicationDirection direction,  
in Security::SecurityFeature feature  
);`
- Description** Not implemented in Orbix SSL/TLS.

## Credentials:invocation\_options\_required Attribute

- IDL** `attribute Security::AssociationOptions  
invocation_options_required;`
- Description** Not implemented in Orbix SSL/TLS.  
Use `SecurityLevel2::QOPPolicy` programmatically or secure invocation policies in the configuration file instead.

---

## Credentials::invocation\_options\_supported Attribute

<b>IDL</b>	attribute <a href="#">Security::AssociationOptions</a> invocation_options_supported;
<b>Description</b>	Not implemented in Orbix SSL/TLS.  Use <code>SecurityLevel2::QOPPolicy</code> programmatically or secure invocation policies in the configuration file instead.

## Credentials::is\_valid()

<b>IDL</b>	boolean is_valid(out <a href="#">Security::UtcT</a> expiry_time);
<b>Description</b>	Returns TRUE if the <code>Credentials</code> object is valid and FALSE otherwise.  Not implemented in Orbix SSL/TLS.

## Credentials::mechanism Attribute

<b>IDL</b>	readonly attribute <a href="#">Security::MechanismType</a> mechanism;
<b>Description</b>	A string, of <a href="#">Security::MechanismType</a> type, that identifies the underlying security mechanism.
<b>Values</b>	Orbix SSL/TLS returns the string 20 which represents SSL/TLS.
<b>See Also</b>	<code>IT_TLS_API::TLS::create_mechanism_policy()</code>

## Credentials::refresh()

<b>IDL</b>	boolean refresh(in any refresh_data);
<b>Description</b>	Not implemented in Orbix SSL/TLS.  Some security mechanisms allow you to extend the expiry time of a <code>Credentials</code> object by refreshing the credentials.

---

## Credentials::set\_attributes()

### IDL

```
boolean set_attributes (  
    in Security::AttributeList requested_attributes,  
    out Security::AttributeList actual_attributes  
);
```

### Description

Not implemented in Orbix SSL/TLS.

# SecurityLevel2::Current Interface

**IDL**

```
// IDL in module SecurityLevel2
local interface Current : SecurityLevel1::Current {
# pragma version Current 1.7
    // Thread specific
    readonly attribute ReceivedCredentials received_credentials;
};
```

**Description** The `Current` object accesses information about the execution context. In Orbix SSL/TLS, the level 2 `Current` interface provides received credentials (originating from a client) to a target object's execution context.

## Current::received\_credentials Attribute

**IDL**

```
readonly attribute ReceivedCredentials received_credentials;
```

At a target object, this thread-specific attribute is the credentials received from a client. They are the credentials of the authenticated principal that made the invocation.

If you have enabled Common Secure Interoperability (CSIv2), the `SecurityLevel2::Current::received_credentials()` operation returns the following credentials type:

- Propagated identity credentials, if present
- Authenticated credentials over the transport, if present and propagated identity credentials are not.
- Transport TLS credentials, if present and the above two are not.

See `IT_CSI::CSIReceivedCredentials` for more details.

**Exceptions** In the case of a pure client, that is, an application that is not servicing an invocation on one of its objects, accessing the `received_credentials` attribute causes a `CORBA::BAD_OPERATION` exception to be raised.





# SecurityLevel2::EstablishTrustPolicy Interface

**IDL**                    `// IDL in module SecurityLevel2  
interface EstablishTrustPolicy : CORBA::Policy {  
    readonly attribute EstablishTrust trust;  
};`

**Description**        A policy of this type can be passed to the `set_policy_overrides()` operation to obtain an object reference that obeys the given trust policy.  
  
The `EstablishTrustPolicy` object has a policy type of [Security::SecEstablishTrustPolicy](#) and is locality constrained.

## EstablishTrustPolicy::trust Attribute

**IDL**                    `readonly attribute EstablishTrust trust;`

**Description**        The `trust` attribute is a structure that contains two members, each stipulating whether trust in the client and trust in the target is enabled.



# SecurityLevel2::InvocationCredentials Policy Interface

**IDL** `// IDL in module SecurityLevel2  
interface InvocationCredentialsPolicy : CORBA::Policy {  
 readonly attribute CredentialsList creds;  
};`

**Description** A policy of this type can be passed to the `set_policy_overrides()` operation to obtain an object reference that uses the given credentials list, `creds`, for operation and attribute invocations.

The `InvocationCredentialsPolicy` object has a policy type of [Security::SecInvocationCredentialsPolicy](#) and is locality constrained.

## **InvocationCredentialsPolicy::creds**

**IDL** `readonly attribute CredentialsList creds;`

**Description** The list of [Credentials](#) objects associated with the `InvocationCredentialsPolicy` object.



# SecurityLevel2::MechanismPolicy Interface

<b>IDL</b>	<pre>// IDL in module <a href="#">SecurityLevel2</a> interface MechanismPolicy : CORBA::Policy {     readonly attribute <a href="#">Security::MechanismTypeList</a> mechanisms; };</pre>
<b>Description</b>	<p>A policy of this type can be passed to the <code>set_policy_overrides()</code> operation to obtain an object reference that uses the specified security mechanisms.</p> <p>The <code>MechanismPolicy</code> object has a policy type of <a href="#">Security::SecMechanismsPolicy</a> and is locality constrained.</p>
<b>See Also</b>	<code>IT_TLS_API::TLS::create_mechanism_policy()</code>

## MechanismPolicy::mechanisms

<b>IDL</b>	<pre>readonly attribute <a href="#">Security::MechanismTypeList</a> mechanisms;</pre>
<b>Description</b>	<p>The mechanisms, in the form of a <a href="#">Security::MechanismTypeList</a>, associated with the <code>MechanismPolicy</code> object.</p>



# SecurityLevel2::PrincipalAuthenticator Interface

## IDL

```
// IDL in module SecurityLevel2
interface PrincipalAuthenticator { // Locality Constrained
#   pragma version PrincipalAuthenticator 1.5

    Security::AuthenticationMethodList
    get_supported_authen_methods(
        in Security::MechanismType mechanism
    );

    Security::AuthenticationStatus authenticate (
        in Security::AuthenticationMethod method,
        in Security::MechanismType mechanism,
        in Security::SecurityName security_name,
        in any auth_data,
        in Security::AttributeList privileges,
        out Credentials creds,
        out any continuation_data,
        out any auth_specific_data
    );

    Security::AuthenticationStatus continue_authentication (
        in any response_data,
        in Credentials creds,
        out any continuation_data,
        out any auth_specific_data
    );
};
```

## Description

This interface provides operations to authenticate a principal and provide it with credentials. For example, the [authenticate\(\)](#) operation is typically called when a user logs on to an application.

---

## PrincipalAuthenticator::authenticate()

**IDL**

```
Security::AuthenticationStatus authenticate (  
    in Security::AuthenticationMethod method,  
    in Security::MechanismType mechanism,  
    in Security::SecurityName security_name,  
    in any auth_data,  
    in Security::AttributeList privileges,  
    out Credentials creds,  
    out any continuation_data,  
    out any auth_specific_data  
);
```

**Description** This operation is called to authenticate the principal. It can also request privilege attributes that the principal requires during its capsule-specific session with the system.

It creates a capsule-specific [Credentials](#) object including the required attributes and is placed on the [SecurityManager](#) object's [own\\_credentials](#) list according to the credential's mechanism type.

In Orbix SSL/TLS, a capsule is effectively identified with an ORB object. The main consequence of this is that credentials are not shared between ORB objects. If you create more than one ORB object in your application, you must call `authenticate()` for each ORB object to make credentials available to both ORBs.

**Return Value** The return value indicates the status of the `creds` parameter:

<code>SecAuthSuccess</code>	A valid <a href="#">Credentials</a> object is available in the <code>creds</code> parameter.
<code>SecAuthFailure</code>	Authentication was in some way inconsistent or erroneous. Credentials have therefore not been created.
<code>SecAuthContinue</code>	The authentication procedure uses a challenge and response mechanism. The <code>creds</code> parameter references a partially initialized <a href="#">Credentials</a> object and the <code>continuation_data</code> indicates details of the challenge.  Not supported by Orbix SSL/TLS.
<code>SecAuthExpired</code>	The authentication data, <code>auth_data</code> , has expired. Credentials have therefore not been created.



---

## Parameters

method	The authentication method to use. For example, <code>IT_TLS_API::IT_TLS_AUTH_METH_PKCS12_FILE</code> . See the <code>IT_TLS_API</code> module for the complete list of authentication methods supported by Orbix SSL/TLS.
mechanism	The security mechanism for creating the returned <a href="#">Credentials</a> object. Leave this parameter blank. It defaults to SSL/TLS.
security_name	The principal's identification information (such as login name). Not used by Orbix SSL/TLS.
auth_data	The principal's authentication information, typically consisting of a certificate, private key and pass phrase. The data inserted into the <code>auth_data</code> parameter depends on the specified authentication method, <code>method</code> .
privileges	The requested privilege attributes. Not supported by Orbix SSL/TLS.
creds	This parameter contains the locality constrained object reference of the newly created <a href="#">Credentials</a> object. It is usable and placed on the <code>Current</code> object's <a href="#">own_credentials</a> list only if the return value is <code>SecAuthSuccess</code> .
continuation_data	Not supported by Orbix SSL/TLS.
auth_specific_data	Not supported by Orbix SSL/TLS.

## PrincipalAuthenticator::continue\_authentication()

**IDL**

```
Security::AuthenticationStatus continue_authentication (  
    in any response_data,  
    in Credentials creds,  
    out any continuation_data,  
    out any auth_specific_data  
);
```

**Description** Not supported by Orbix SSL/TLS.

---

## PrincipalAuthenticator::get\_supported\_authen\_methods()

### IDL

```
Security::AuthenticationMethodList  
get_supported_authen_methods(  
    in Security::MechanismType mechanism  
);
```

### Description

Not implemented in Orbix SSL/TLS.

# SecurityLevel2::QOPPolicy Interface

**IDL**            `// IDL in module SecurityLevel2  
interface QOPPolicy : CORBA::Policy {  
    readonly attribute Security::QOP qop;  
};`

**Description**    A QOP policy object can be passed to the `set_policy_overrides()` operation to obtain an object reference that uses the specified quality of protection policy.

**See Also**        [Security::SecQOPPolicy](#)

## QOPPolicy::qop Attribute

**IDL**            `readonly attribute Security::QOP qop;`

**Description**    The quality of protection, of [Security::QOP](#) enumeration type, associated with the QOPPolicy object.



# SecurityLevel2::ReceivedCredentials Interface

**IDL**

```
// IDL in module SecurityLevel2
interface ReceivedCredentials : Credentials {
# pragma version ReceivedCredentials 1.5
    readonly attribute Credentials accepting_credentials;

    readonly attribute Security::AssociationOptions
        association_options_used;

    readonly attribute Security::DelegationState delegation_state;

    readonly attribute Security::DelegationMode delegation_mode;
};
```

**Description** A ReceivedCredentials object stores the security attributes of a remote client. It is made available in an execution context on the server side and can be obtained from a [SecurityLevel2::Current](#) object.

**See Also** [SecurityLevel2::Current](#)  
[IT\\_TLS\\_API::TLSReceivedCredentials](#)

## ReceivedCredentials::accepting\_credentials Attribute

**IDL** readonly attribute [Credentials](#) accepting\_credentials;

**Description** Not implemented in Orbix SSL/TLS.

## ReceivedCredentials::association\_options\_used Attribute

**IDL** readonly attribute [Security::AssociationOptions](#) association\_options\_used;

**Description** Not implemented in Orbix SSL/TLS.

---

### ReceivedCredentials::delegation\_mode Attribute

**IDL**            readonly attribute [Security](#)::DelegationMode delegation\_mode;

**Description**    Not implemented in Orbix SSL/TLS.

### ReceivedCredentials::delegation\_state Attribute

**IDL**            readonly attribute [Security](#)::DelegationState delegation\_state;

**Description**    Not implemented in Orbix SSL/TLS.

# SecurityLevel2::SecurityManager Interface

**IDL**

```
// IDL in module SecurityLevel2
interface SecurityManager {
    readonly attribute Security::MechandOptionsList
        supported_mechanisms;
    readonly attribute CredentialsList own_credentials;
    readonly attribute RequiredRights required_rights_object;
    readonly attribute PrincipalAuthenticator
        principal_authenticator;

    readonly attribute AccessDecision access_decision;
    readonly attribute AuditDecision audit_decision;

    TargetCredentials get_target_credentials (in Object obj_ref);

    void remove_own_credentials(in Credentials creds);

    CORBA::Policy get_security_policy (
        in CORBA::PolicyType policy_type
    );
};
```

**Description** In Orbix SSL/TLS, this class is used to access ORB-specific information.

## SecurityManager::access\_decision Attribute

**IDL** readonly attribute AccessDecision access\_decision;

**Description** Not implemented in Orbix SSL/TLS.

## SecurityManager::audit\_decision Attribute

**IDL** readonly attribute AuditDecision audit\_decision;

---

**Description** Not implemented in Orbix SSL/TLS.

### **SecurityManager::get\_security\_policy()**

**IDL**

```
CORBA::Policy get_security_policy (  
    in CORBA::PolicyType policy_type  
);
```

**Description** Not implemented in Orbix SSL/TLS.

### **SecurityManager::get\_target\_credentials()**

**IDL**

```
TargetCredentials get_target_credentials(  
    in Object target;  
);
```

**Description** Returns the target credentials for an object referenced by the specified object reference, `target`. For example, this operation is typically used on the client side to obtain the target credentials for a remote object.

#### **Parameters**

`target` An object reference.

### **SecurityManager::own\_credentials Attribute**

**IDL**

```
readonly attribute CredentialsList own_credentials;
```

**Description** Holds an application's own credentials, which are established by calling [authenticate\(\)](#) on the application's own [PrincipalAuthenticator](#) object.

### **SecurityManager::principal\_authenticator Attribute**

**IDL**

```
readonly attribute PrincipalAuthenticator principal_authenticator;
```

**Description** Holds a reference to the [PrincipalAuthenticator](#) object that can be used by the application to authenticate principals and obtain credentials.



---

## SecurityManager::remove\_own\_credentials()

**IDL**

```
void remove_own_credentials(  
    in Credentials creds  
);
```

**Description**

Removes credentials that were put on the `own_credentials` list using the [PrincipalAuthenticator](#). This operation does not manipulate or destroy the objects in any way.

**Parameters**

`creds`            The [Credentials](#) object to be removed from the list.

## SecurityManager::required\_rights\_object Attribute

**IDL**

```
readonly attribute RequiredRights required_rights_object;
```

**Description**

Not implemented in Orbix SSL/TLS.

## SecurityManager::supported\_mechanisms Attribute

**IDL**

```
readonly attribute Security::MechandOptionsList  
supported_mechanisms;
```

**Description**

Not implemented in Orbix SSL/TLS.



# SecurityLevel2::TargetCredentials Interface

**IDL**

```
// IDL in module SecurityLevel2
interface TargetCredentials : Credentials {
    readonly attribute Credentials
        initiating_credentials;

    readonly attribute Security::AssociationOptions
        association_options_used;
};
```

**Description** A `TargetCredentials` object holds the security attributes of an authenticated target object. To obtain the target credentials for a remote object, call the [SecurityManager::get\\_target\\_credentials\(\)](#) operation.

**See Also** `IT_TLS_API::TLSTargetCredentials`

## TargetCredentials::association\_options\_used Attribute

**IDL**

```
readonly attribute Security::AssociationOptions
    association_options_used;
```

**Description** Not implemented in Orbix SSL/TLS.

## TargetCredentials::initiating\_credentials Attribute

**IDL**

```
readonly attribute Credentials initiating_credentials;
```

**Description** Not implemented in Orbix SSL/TLS.



# Appendix A

## System Exceptions

This appendix defines the system exceptions returned by Orbix.

BAD_CONTEXT	This exception is raised if a client invokes an operation but the passed context does not contain the context values required by the operation.
BAD_INV_ORDER	This exception indicates that the caller has invoked operations in the wrong order. For example, it can be raised by an ORB if an application makes an ORB-related call without having correctly initialized the ORB first.
BAD_OPERATION	This exception indicates that an object reference denotes an existing object, but that the object does not support the operation that was invoked.
BAD_PARAM	<p>This exception is raised if a parameter passed to a call is out of range or otherwise considered illegal. For example, an ORB may raise this exception if null values or null pointers are passed to an operation (for language mappings where the concept of a null pointers or null values applies).</p> <p>BAD_PARAM can also be raised as a result of client generating requests with incorrect parameters using the DII.</p>
BAD_TYPECODE	This exception is raised if the ORB encounters a malformed type code (for example, a type code with an invalid <code>TCKind</code> value).
COMM_FAILURE	This exception is raised if communication is lost while an operation is in progress, after the request was sent by the client, but before the reply from the server has been returned to the client.

## Appendix A

---

DATA_CONVERSION	This exception is raised if an ORB cannot convert the representation of data as marshaled into its native representation or vice-versa. For example, DATA_CONVERSION can be raised if wide character codeset conversion fails, or if an ORB cannot convert floating point values between different representations.
FREE_MEM	This exception is raised if the ORB failed in an attempt to free dynamic memory. For example, it is raised because of heap corruption or memory segments being locked.
IMP_LIMIT	This exception indicates that an implementation limit was exceeded in the ORB run time. For example, an ORB may reach the maximum number of references it can hold simultaneously in an address space, the size of a parameter may have exceeded the allowed maximum, or an ORB may impose a maximum on the number of clients or servers that can run simultaneously.
INITIALIZE	This exception is raised if an ORB encounters a failure during its initialization, such as failure to acquire networking resources or detection of a configuration error.
INTERNAL	This exception indicates an internal failure in an ORB. For example, it is raised if an ORB detected corruption of its internal data structures.
INTF_REPOS	This exception is raised if an ORB cannot reach the interface repository, or some other failure relating to the interface repository is detected.
INV_FLAG	This exception indicates that an invalid flag was passed to an operation. For example, it is raised when creating a DII request.

---

INV_IDENT	This exception indicates that an IDL identifier is syntactically invalid. For example it may be raised if an identifier passed to the interface repository does not conform to IDL identifier syntax, or if an illegal operation name is used with the DII.
INV_OBJREF	<p>This exception indicates that an object reference is internally malformed. For example, the repository ID may have incorrect syntax or the addressing information may be invalid. This exception is raised by <code>ORB::string_to_object</code> if the passed string does not decode correctly.</p> <p>An ORB implementation might detect calls via nil references (although it is not obliged to detect them). <code>INV_OBJREF</code> is used to indicate this.</p>
INV_POLICY	This exception is raised when an invocation cannot be made due to an incompatibility between policy overrides that apply to the particular invocation.
INVALID_TRANSACTION	This exception indicates that the request carried an invalid transaction context. For example, this exception could be raised if an error occurred when trying to register a resource.
MARSHAL	<p>This exception is raised if a request or reply from the network is structurally invalid. This error typically indicates a bug in either the client-side or server-side run time. For example, if a reply from the server indicates that the message contains 1000 bytes, but the actual message is shorter or longer than 1000 bytes, the ORB raises this exception.</p> <p><code>MARSHAL</code> can also be caused by using the DII or DSI incorrectly. For example, it is raised if the type of the actual parameters sent does not agree with IDL signature of an operation.</p>

## Appendix A

---

NO_IMPLEMENT	This exception is raised if the operation that was invoked exists (it has an IDL definition) but no implementation for that operation exists. For example, NO_IMPLEMENT can be raised by an ORB if a client asks for an object's type definition from the interface repository, but no interface repository is provided by the ORB.
NO_MEMORY	This exception indicates that the ORB run time has run out of memory.
NO_PERMISSION	This exception is raised if an invocation fails because the caller has insufficient privileges.
NO_RESOURCES	This exception indicates that the ORB has encountered some general resource limitation. For example, the run time may have reached the maximum permissible number of open connections.
NO_RESPONSE	This exception is raised if a client attempts to retrieve the result of a deferred synchronous call but the response for the request is not yet available.
OBJ_ADAPTER	This exception typically indicates an administrative mismatch. For example, a server may have made an attempt to register itself with an implementation repository under a name that is already in use, or a name that is unknown to the repository. OBJ_ADAPTER is also raised by the POA to indicate problems with application-supplied servant managers.



---

OBJECT\_NOT\_EXIST

This exception is raised whenever an invocation on a deleted object is performed. It is an authoritative “hard” fault report. Anyone receiving it is allowed (even expected) to delete all copies of this object reference and to perform other appropriate “final recovery” style procedures.

Bridges forward this exception to clients, also destroying any records they may hold (for example, proxy objects used in reference translation). The clients could in turn purge any of their own data structures.

PERSIST\_STORE

This exception indicates a persistent storage failure. For example, it is raised if there is a failure to establish a database connection or corruption of a database.

REBIND

This exception is raised when the current effective `RebindPolicy` has a value of `NO_REBIND` or `NO_RECONNECT` and an invocation on a bound object reference results in a `LocateReply` message with status `OBJECT_FORWARD` or a `Reply` message with status `LOCATION_FORWARD`. This exception is also raised if the current effective `RebindPolicy` has a value of `NO_RECONNECT` and a connection must be re-opened. The invocation can be retried once the effective `RebindPolicy` is changed to `TRANSPARENT` or binding is re-established through an invocation of `CORBA::Object::validate_connection()`.

TRANSACTION\_REQUIRED

This exception indicates that the request carried a null transaction context, but an active transaction is required.

TRANSACTION\_ROLLEDBACK

This exception indicates that the transaction associated with the request has already been rolled back or marked to roll back. The requested operation either could not be performed or was not performed because further computation on behalf of the transaction would be fruitless.

## Appendix A

---

TRANSIENT

This exception indicates that the ORB attempted to reach an object and failed. It is not an indication that an object does not exist. Instead, it simply means that no further determination of an object's status was possible because it could not be reached. For example, this exception is raised if an attempt to establish a connection fails because the server or the implementation repository is down.

UNKNOWN

This exception is raised if an operation implementation throws a non-CORBA exception (such as an exception specific to the implementation's programming language), or if an operation raises a user exception that does not appear in the operation's `raises` expression.

UNKNOWN is also raised if the server returns a system exception that is unknown to the client. (This can happen if the server uses a later version of CORBA than the client and new system exceptions have been added to the later version.)

# Index

## Symbols

() Subscript Operators 242, 286

## A

absolute\_name Attribute 54

abstract\_base\_values Attribute 272

access Attribute 283

activate() 1210

activate\_object() 1189

activate\_object\_with\_id() 1189

active\_groups() method 930

AdapterActivator class 1169

adapter\_id attribute 1152

AdapterInactive exception 1210

ADAPTS 570

add() 85, 135, 158

add\_client\_request\_interceptor() 1134

add\_consume() 135

add\_constraints() 465

add\_filter() 471

add\_in\_arg() 225

add\_inout\_arg() 225

add\_ior\_component() 1125

add\_ior\_component\_to\_profile() 1126

add\_ior\_interceptor() 1134

add\_item() 158

add\_item\_consume() 158

add\_link() 512

add\_listener() 814

add\_mapping\_constraints() 479

add\_member method 922

add\_named\_in\_arg() 226

add\_named\_inout\_arg() 226

add\_named\_out\_arg() 226

add\_out\_arg() 226

\_add\_ref() 265, 281, 829, 831

add\_ref() 8

add\_reply\_service\_context() 1152

add\_request\_service\_context() 1110

Address data type 338

add\_server\_request\_interceptor() 1135

add\_type() 558

add\_value() 158

add\_value\_consume() 159

admin\_if attribute 545

AdminLimitExceeded exception 360

AdminNotFound exception 359

AdminPropertiesAdmin

    get\_admin() 349

AliasDef Interface 29

allocate\_slot\_id() 1135

AlreadyBound exception 325

AlreadyMasked exception 556

Any Class 31

any IDL type 31

AnySeq Sequence 692

ApplicationId data type 931

arguments() 226

arguments attribute 1135, 1144

ArrayDef Interface 41

assign() 701

attach\_callback() 469

AttributeDef Interface 43

AttributeDescription Structure 8

AttributeMode Enumeration 9

audience xxvii

authenticate() 1266

authentication state 1252

AUTOMATIC 841

AutomaticWorkQueue 1049

    high\_water\_mark 1050

    low\_water\_mark 1050

    shutdown() 1050

AutomaticWorkQueueFactory 1051

    create\_work\_queue() 1051

    create\_work\_queue\_with\_thread\_stack\_size() 1052

AVA

    convert() 787

AVA interface 787

AVAList

    convert() 792

    get\_ava\_by\_oid() 793

    get\_ava\_by\_oidtag() 793

    get\_num\_avas() 793

interface 791

## B

BAD\_CONTEXT exception 1279  
 BAD\_INV\_ORDER exception 1279  
 BadKind Exception 249  
 BAD\_OPERATION exception 1279  
 BAD\_PARAM exception 1279  
 BAD\_TYPECODE exception 1279  
 base\_interfaces Attribute 144  
 base\_value Attribute 273  
 before\_completion() 604  
 begin() 590  
 bind() 325  
 bind\_context() 326  
 BindingIterator interface 321  
 BindingList sequence 318  
 Binding structure 317  
 BindingType enumeration 318  
 bind\_new\_context() 327  
 bind\_object\_group() method 965  
 boolean\_changed() 822  
 bound Attribute 233, 245, 289  
 Bounds Exception 250  
 byte\_order() 1086

## C

CallbackNotFound exception 461  
 CannotProceed exception 328  
 CertConstraintsPolicy 1023  
 CertValidatorPolicy 1025  
 channel manager 1099, 1101  
 ChannelManager::create() 1101  
 ChannelManager::createTyped() 1103  
 ChannelManager::find() 1102  
 ChannelManager::findByRef() 1102  
 ChannelManager::findTyped() 1103  
 ChannelManager::findTypedByRef() 1103  
 ChannelManager::list() 1102  
 ChannelManager::listTyped() 1104  
 char\*() 241, 285  
 clear() 129  
 clear\_filter() 940  
 ClientRequestInfo interface 1107  
 ClientRequestInterceptor interface 1115  
 codec\_factory attribute 1136  
 COMM\_FAILURE exception 1279  
 commit() 590, 599, 605

commit\_on\_completion\_of\_next\_call() 843  
 commit\_one\_phase() 599  
 commit\_subtransaction() 602  
 Common CORBA Data Types 8  
 component\_count() 702  
 concrete\_base\_type() 250  
 ConfigList sequence 810  
 config\_scope() 833  
 configuration context 809  
 configuration domain 809  
 Configuration interface 813  
 configuration scope 809  
 connect\_any\_pull\_consumer() 386  
 connect\_any\_pull\_supplier() 384  
 connect\_any\_push\_consumer() 390  
 connect\_any\_push\_supplier() 388  
 connect\_group\_any\_push\_consumer() 969, 970  
 connect\_group\_sequence\_push\_consumer() 973  
 connect\_group\_structured\_push\_consumer() 977  
 ConnectionAlreadyActive exception 359  
 ConnectionAlreadyInactive exception 359  
 connect\_sequence\_pull\_consumer() 402  
 connect\_sequence\_pull\_supplier() 398  
 connect\_sequence\_push\_consumer() 404, 974  
 connect\_sequence\_push\_supplier() 400  
 connect\_structured\_pull\_consumer() 410  
 connect\_structured\_pull\_supplier() 408  
 connect\_structured\_push\_consumer() 414, 978  
 connect\_structured\_push\_supplier() 412  
 ConstantDef Interface 47  
 ConstantDescription Structure 9  
 Constraint 487  
 constraint\_grammar 464  
 ConstraintRecipe 530  
 ConstructionPolicy Interface 51  
 ConsumerAdmin interface 293, 361  
 Contained Interface 53  
 Container Interface 59  
 containing\_repository Attribute 54  
 contents() 62, 273  
 content\_type() 251  
 Context\_Class 79  
 context in configuration 809  
 ContextList 689  
 ContextList Class 85  
 context\_name() 79  
 contexts() 227  
 contexts Attribute 171  
 contexts attribute 1144

- Control class 575
- conventions in document xxviii
- convert() 787, 792, 797, 799
- Cookie Native Type 1223
- Coordinator class 577
- copy() 208, 703
- \_copy\_value() 265
- CORBA 2.3 specification xxvii
- CosEventChannelAdmin::EventChannel Interface 295
- CosEventChannelAdmin::SupplierAdmin interface 305
- CosEventChannelAdmin module 291
- CosEventComm::Disconnected 307
- CosEventComm::PushConsumer Interface 313
- CosEventComm::PushSupplier Interface 315
- CosEventCom module 307
- CosNaming module 317
- CosNotification
  - AdminProperties 344
  - AdminPropertiesAdmin Interface 349
  - EventBatch 342
  - EventTypeSeq 342
  - NamedPropertyRangeSeq 346
  - PropertyErrorSeq 345
  - PropertyName 344
  - PropertySeq 344
  - PropertyValue 344
  - QoSAdmin Interface 351
  - QoSProperties 344
  - StructuredEvent 341
  - UnsupportedAdmin 347
  - UnsupprtedQoS 346
- CosNotifyChannelAdmin
  - AdminID 357
  - AdminIDSeq 358
  - ChannelID 358
  - ChannelIDSeq 358
  - ObtainInfoMode 356
  - ProxyID 356
  - ProxyIDSeq 356
  - ProxyType 355
  - AdminLimit 358
  - ChannelNotFound exception 360
  - ClientType 356
- CosNotifyFilter
  - CallbackID 460
  - CallbackIDSeq 460
  - ConstraintExp 457
  - ConstraintExpSeq 458
  - ConstraintID 457
  - ConstraintIDSeq 458
  - ConstraintInfo 458
  - ConstraintInfoSeq 458
  - InterFilterGroupOperator 357
  - InvalidValue exception 461
  - MapingConstraintPair 459
  - MappingConstraintInfo 459
  - MappingConstraintInfoSeq 459
  - MappingConstraintPairSeq 459
- CosTrading 487
  - Admin 497
  - LinkAttributes 515
  - Lookup 517
- CosTradingDynamic
  - DPEvalFailure exception 548
  - DynamicProp Struct 547
- CosTradingDynamic Module 547
- CosTradingRepos Module 551
- CosTransactions, data types 568
- CosTransactions module 563
- CostTypedEventChannelAdmin::Key Type 612
- CostTypedEventChannelAdmin::InterfaceNotSupported 611
- CostTypedEventChannelAdmin::NoSuchImplementation 611
- CostTypedEventChannelAdmin::TypedConsumerAdmin Interface 613
- CostTypedEventChannelAdmin::TypedEventChannel Interface 615
- CostTypedEventChannelAdmin::TypedProxyPushConsumer Interface 617
- CostTypedEventChannelAdmin::TypedSupplierAdmin Interface 619
- CostTypedEventChannelAdmin module 611
- CostTypedEventComm::TypedPushConsumer Interface 623
- CostTypedEventComm module 621
- count() 86, 135, 157
- create() 610
- create\_active() method 929
- create\_alias() 63
- create\_alias\_tc() 183

- create\_array() 218
  - create\_array\_tc() 184
  - create\_attribute() 145, 274
  - create\_channel() 375
  - create\_child() 80
  - create\_constant() 64
  - create\_context\_list() 184
  - create\_dyn\_any() 732
  - create\_dyn\_any\_from\_type\_code() 732
  - create\_enum() 65
  - create\_enum\_tc() 185
  - create\_environment() 185
  - create\_exception() 66
  - create\_exception\_list() 185
  - create\_exception\_tc() 186
  - create\_filter() 475
  - create\_fixed() 219
  - create\_fixed\_tc() 186
  - create\_id\_assignment\_policy() 1190
  - create\_id\_uniqueness\_policy() 1191
  - create\_implicit\_activation\_policy() 1192
  - create\_interface() 67
  - create\_interface\_tc() 186
  - create\_lifespan\_policy() 1192
  - create\_list() 187
  - create\_mapping\_filter() 476
  - create\_module() 68
  - create\_named\_value() 187
  - create\_native() 69
  - create\_native\_tc() 188
  - create\_operation() 146, 275
  - create\_operation\_list() 188
  - create\_POA() 1193
  - create\_policy() 189, 1141
  - create\_random() method 928
  - create\_recursive\_tc() 189
  - create\_reference() 1195
  - create\_reference\_with\_id() 1196
  - \_create\_request() 163
  - create\_request\_processing\_policy() 1196
  - create\_round\_robin() method 928
  - create\_sequence() 220
  - create\_sequence\_tc() 190
  - create\_servant\_retention\_policy() 1198
  - create\_string() 220
  - create\_string\_tc() 191
  - create\_struct() 70
  - create\_struct\_tc() 191
  - create\_subtransaction() 579
  - create\_thread\_policy() 1198
  - create\_union() 71
  - create\_union\_tc() 191
  - create\_value() 73
  - create\_value\_box() 75
  - create\_value\_box\_tc() 192
  - create\_value\_member() 276
  - create\_value\_tc() 192
  - create\_wstring() 221
  - create\_wstring\_tc() 193
  - Credentials
    - authentication\_state 1252
    - destroy() 1252, 1253, 1254, 1255, 1256, 1271, 1272, 1277
    - get\_attributes() 1254
  - Credentials interface 1251, 1271, 1277
  - ctx() 227, 235
  - Current
    - received\_credentials 1247, 1257, 1273, 1274, 1275
    - received\_credentials attribute 1247, 1257
  - Current Class 1173
  - Current class 589
  - current\_component() 703
  - Current Interface 87, 1274, 1275
  - Current interface 1121, 1247, 1257
  - current\_member\_kind() 752, 764
  - current\_member\_name() 752, 764
  - CustomMarshal Value Type 89
- D**
- DATA\_CONVERSION exception 1280
  - DataInputStream Value Type 93
  - DataOutputStream Value Type 109
  - deactivate() 1211
  - deactivate\_object() 1199
  - DefaultFollowTooPermissive exception 510
  - default\_index() 251
  - \_DEFAULT in logging 934
  - default\_value 479
  - def\_follow\_policy attribute 505
  - def\_hop\_count attribute 505
  - defined\_in Attribute 54
  - DefinitionKind Enumeration 10
  - def\_kind Attribute 149
  - def\_match\_card attribute 505
  - def\_return\_card attribute 505
  - def\_search\_card attribute 506
  - delete\_values() 81

- describe() 29, 43, 47, 55, 127, 133, 147, 151, 172, 247, 261, 263, 277, 538
- describe\_contents() 76
- describe\_interface() 147
- describe\_link() 513
- describe\_type() 559
- describe\_value() 278
- Description Structure 55
- destroy() 149, 193, 209, 321, 329, 367, 422, 468, 483, 525, 704, 1200, 1252, 1253, 1254, 1255, 1256, 1271, 1272, 1277
- destroy() method 926
- \_destroy\_this() 829, 831
- digits Attribute 137
- DII and DSI Quick Reference 4
- discard\_requests() 1211
- disconnect\_push\_consumer() 985, 986
- disconnect\_sequence\_pull\_consumer() 439
- disconnect\_sequence\_pull\_supplier() 443
- disconnect\_sequence\_push\_consumer() 446, 987, 988
- disconnect\_sequence\_push\_supplier() 447
- disconnect\_structured\_pull\_consumer() 449
- disconnect\_structured\_pull\_supplier() 452
- disconnect\_structured\_push\_consumer() 454, 989, 990
- disconnect\_structured\_push\_supplier() 455
- discriminator\_kind() 758
- discriminator\_type() 251
- discriminator\_type Attribute 263
- discriminator\_type\_def Attribute 264
- documentation, other xxviii
- domain, configuration 809
- DomainManager Interface 125
- double\_changed() 823
- \_downcast() 266, 281, 1086
- DsEventLog Module 635
- DsLogAdmin
  - TimeT 646
  - :LogFullActionType 648
- DsLogAdmin::AdministrativeState 651
- DsLogAdmin::Anys 648
- DsLogAdmin::AvailabilityStatus 648
- DsLogAdmin::BasicLog 653
- DsLogAdmin::BasicLogFactory 655
- DsLogAdmin::CapacityAlarmThresholdList 650
- DsLogAdmin::Constraint 646
- DsLogAdmin::DaysOfWeek 649
- DsLogAdmin::ForwardingState 651
- DsLogAdmin::IntervalsOfDay 649
- DsLogAdmin::InvalidAttribute 644
- DsLogAdmin::InvalidConstraint 642
- DsLogAdmin::InvalidGrammar 642
- DsLogAdmin::InvalidLogFullAction 644
- DsLogAdmin::InvalidMask 642
- DsLogAdmin::InvalidParam 641
- DsLogAdmin::InvalidRecordId 643
- DsLogAdmin::InvalidThreshold 641
- DsLogAdmin::InvalidTime 641
- DsLogAdmin::InvalidTimeInterval 642
- DsLogAdmin::Iterator 657
- DsLogAdmin::Log 659
- DsLogAdmin::LogDisabled 643
- DsLogAdmin::LogFull 643
- DsLogAdmin::LogId 645
- DsLogAdmin::LogIdList 652
- DsLogAdmin::LogList 651
- DsLogAdmin::LogLocked 643
- DsLogAdmin::LogMgr 675
- DsLogAdmin::LogOffDuty 643
- DsLogAdmin::LogRecord 647
- DsLogAdmin::NVList 646
- DsLogAdmin::NVPair 646
- DsLogAdmin::OperationalState 651
- DsLogAdmin::QoSList 652
- DsLogAdmin::QoSType 652
- DsLogAdmin::RecordId 645
- DsLogAdmin::RecordIdList 646
- DsLogAdmin::RecordList 647
- DsLogAdmin::Threshold 650
- DsLogAdmin::Time24 648
- DsLogAdmin::Time24Interval 649
- DsLogAdmin::TimeInterval 647
- DsLogAdmin::UnsupportedQoS 644
- DsLogAdmin::WeekMask 650
- DsLogAdmin::WeekMaskItem 649
- DsLogAdmin Module 641
- DsLogNotification::AttributeType 678
- DsLogNotification::AttributeValueChange 679
- DsLogNotification::ObjectCreation 678
- DsLogNotification::ObjectDeletion 678
- DsLogNotification::PerceivedSeverityType 677
- DsLogNotification::ProcessingErrorAlarm 680
- DsLogNotification::StateChange 680
- DsLogNotification::StateType 679
- DsLogNotification::ThresholdAlarm 677
- DsNotifyLogAdmin::NotifyLogFactory Interface 685

DsNotifyLogAdmin::NotifyLog Interface 683  
 DsNotifyLogAdmin Module 677, 681  
   duplicate() 164, 194  
 DuplicateGroup exception 919  
 DuplicateLinkName exception 511  
 DuplicateMember exception 919  
 DuplicateName exception 1136  
 DuplicatePolicyName exception 492  
 DuplicatePropertyName 493  
 DuplicateServiceTypeName exception 556  
 DynamicImplementation class 1175  
 Dynamic module 689  
 DynamicPropEval 549  
 DynAny Class 695  
 DynAnyFactory Class 731  
 DynArray Class 735  
 DynEnum Class 739  
 DynFixed Class 743  
 DynSequence Class 745  
 DynStruct Class 751  
 DynUnion Class 757  
 DynValue Class 763

**E**

effective\_profile attribute 1110  
 effective\_target attribute 1110  
 EITHER 571  
 element\_type Attribute 41, 233  
 element\_type\_def Attribute 41, 234  
 EnumDef Interface 127  
 env() 227  
 Environment Class 129  
 equal() 252, 704  
 equivalent() 252  
 establish\_components() 1129  
 EstablishTrus Policy 1259  
 etherealize() 1217  
 evalDP() 549  
 EventChannel::destroy() 295  
 EventChannel::for\_consumers() 295  
 EventChannel::for\_suppliers() 295  
 EventChannelFactory::create\_channel() 897  
 EventChannelFactory::find\_channel() 898  
 EventChannelFactory::find\_channel\_by\_id() 898  
 EventChannelFactory::list\_channels() 898  
 EventChannelFactory interface 375  
 EventChannel interface 295, 369  
 EventId data type 932  
 EventLog 637

EventLogFactory 639  
 EventLogFactory::create() 639  
 EventLogFactory::create\_with\_id() 640  
 EventLog Interface 939  
 EventParameters data type 932  
 EventPriority data type 932  
 except() 236  
 exception() 130  
 Exception Class 131  
 ExceptionDef Interface 133  
 ExceptionDescription Structure 10  
   ~ExceptionHandler() 1086  
 ExceptionHolder() constructors 1086  
 ExceptionHolder value type 1085  
 ExceptionList 689  
 ExceptionList Class 135  
 exceptions 564  
 exceptions() 227  
 exceptions, system 1279  
 exceptions Attribute 172  
 exceptions attribute 1144  
 expand\_filter() 941  
 export() 538  
 export\_proxy() 532  
 Extension  
   convert() 797  
   get\_extension\_by\_oid() 801  
   get\_extension\_by\_oidtag() 801  
 Extension interface 797  
 ExtensionList  
   convert() 799  
   get\_num\_extensions() 802  
 ExtensionList interface 799

**F**

FilterAdmin interface 471  
 FilterFactory interface 475  
 FilterID Data Type 458  
 FilterIDSeq Data Type 458  
 FilterNotFound exception 461  
 filters  
   IDL 463  
 find\_group() method 929  
 find\_POA() 1201  
 FixedDef Interface 137  
 fixed\_digits() 253  
 fixed\_scale() 253  
 flags() 153  
 FollowOption 491



FORBIDS 570  
 forget() 599  
 format\_message() 933  
 forward\_reference attribute 1145  
 ForwardRequest exception 1105, 1162  
 FPS\_POLICY\_BASE 901  
 FREE\_MEM exception 1280  
 from\_any() 705  
 FullInterfaceDescription Structure 147  
 FullValueDescription Structure 278  
 fully\_describe\_type() 560

**G**

get\_all\_channels() 376  
 get\_all\_constraints() 467  
 get\_all\_consumeradmins() 374  
 get\_all\_filters() 472  
 get\_all\_mapping\_constraints 482  
 get\_all\_supplieradmins() 374  
 get\_as\_string() 740  
 get\_as\_ulong() 740  
 get\_attributes() 1254  
 get\_ava\_by\_oid() 793  
 get\_ava\_by\_oidtag() 793  
 get\_boolean() 706, 815  
 get\_callbacks() 470  
 get\_canonical\_typecode() 222  
 get\_char() 706  
 get\_compact\_typecode() 253  
 get\_constraints() 467  
 get\_consumeradmin() 373  
 get\_control() 591  
 get\_coordinator() 576  
 get\_default\_context() 194  
 get\_der\_serial\_number() 806  
 get\_discriminator() 758  
 \_get\_domain\_managers() 165  
 get\_domain\_policy() 126  
 get\_double() 707, 816  
 get\_dyn\_any() 707  
 get\_effective\_component() 1111  
 get\_effective\_components() 1111  
 get\_effective\_policy() 1126  
 get\_elements() 736, 746  
 get\_elements\_as\_dyn\_any() 736, 746  
 get\_event\_channel() 377  
 get\_exception() 1087  
 get\_exception\_with\_list() 1087  
 get\_extension\_by\_oid() 801  
 get\_extension\_by\_oidtag() 801  
 get\_filter() 472, 941  
 get\_float() 708  
 \_get\_interface() 165  
 get\_length() 746  
 get\_list() 816  
 get\_long() 708, 818  
 get\_longlong() 709  
 get\_mapping\_constraints() 482  
 get\_member() method 923  
 get\_member\_load() method 925  
 get\_members() 753, 765  
 get\_members\_as\_dyn\_any() 753, 765  
 get\_member\_timeout() method 926  
 get\_next\_response() 195  
 get\_num\_avas() 793  
 get\_num\_extensions() 802  
 get\_object\_id() 1173  
 get\_octet() 709  
 get\_parent\_status() 579  
 get\_POA() 1174  
 \_get\_policy() 165  
 \_get\_policy\_overrides() 166  
 get\_policy\_overrides() 213  
 get\_primitive() 222  
 get\_proxy\_consumer() 420  
 get\_proxy\_supplier() 365  
 get\_reference() 710  
 get\_reply\_service\_context() 1145  
 get\_request\_policy() 1112  
 get\_request\_service\_context() 1146  
 get\_response() 228  
 get\_servant() 1201  
 get\_servant\_manager() 1202  
 get\_server\_policy() 1152  
 get\_service\_information() 195  
 get\_short() 710  
 get\_slot() 1121, 1146  
 get\_state() 1212  
 get\_status() 580, 591  
 get\_string() 710, 818  
 get\_supplieradmin() 373  
 get\_target\_credentials() 1273, 1274, 1275  
 get\_terminator() 576  
 get\_timeout() 592  
 get\_top\_level\_status() 580  
 get\_transaction\_name() 581, 592  
 get\_txcontext() 581  
 get\_typecode() 711

get\_ulong() 711  
 get\_ulonglong() 712  
 get\_ushort() 712  
 get\_val() 713  
 get\_value() 744  
 get\_values() 81  
 get\_wchar() 713  
 get\_wstring() 714  
 GroupId data type 918  
 GroupList data type 919  
 GroupNotifyPublish 983  
 GroupProxyPushSupplier 969  
 GroupPushConsumer 985  
 GroupSequenceProxyPushSupplier 973  
 GroupSequencePushConsumer 987  
 GroupStructuredProxyPushSupplier 977  
 GroupStructuredPushConsumer 989

## H

hash\_top\_level\_tran() 581  
 hash\_transaction() 582  
 has\_no\_active\_member() 759  
 HasSubTypes exception 556  
 HeuristicCommit exception 565  
 HeuristicHazard exception 565  
 HeuristicMixed exception 565  
 HeuristicRollback exception 565  
 hold\_requests() 1213  
 HowManyProps 518

## I

id() 254  
 IdAssignmentPolicy class 1177  
 ID\_ASSIGNMENT\_POLICY\_ID constant 1162  
 IdAssignmentPolicyValue enumeration 1163  
 id Attribute 56  
 Identifier Alias 553  
 IDLType Interface 141  
 id\_to\_reference() 1202  
 id\_to\_servant() 1203  
 IdUniquenessPolicy class 1179  
 ID\_UNIQUENESS\_POLICY\_ID constant 1163  
 IdUniquenessPolicyValue enumeration 1163  
 IllegalConstraint exception 493  
 IllegalLinkName exception 511  
 IllegalOfferId exception 493  
 IllegalPolicyName 519  
 IllegalPreference 519

IllegalPropertyName exception 493  
 IllegalRecipie exception 531  
 IllegalServiceType exception 493  
 IllegalTraderName exception 535  
 IMPLICIT\_ACTIVATION 1164  
 ImplicitActivationPolicy class 1181  
 IMPLICIT\_ACTIVATION\_POLICY\_ID constant 1164  
 ImplicitActivationPolicyValue enumeration 1164  
 IMP\_LIMIT exception 1280  
 in() 241, 285  
 Inactive exception 566  
 incarnate() 1218  
 incarnation 558  
 IncarnationNumber 554  
 InconsistentTypeCode User Exception Class 734  
 INITIALIZE exception 1280  
 initializers Attribute 279  
 Initializer Structure 11  
 inout() 241, 285  
 insert\_any() 714  
 insert\_boolean() 715  
 insert\_char() 716  
 insert\_double() 716  
 insert\_dyn\_any() 717  
 insert\_float() 717  
 insert\_long() 718  
 insert\_longlong() 719  
 insert\_octet() 719  
 insert\_reference() 720  
 insert\_short() 720  
 insert\_string() 721  
 insert\_typecode() 722  
 insert\_ulong() 722  
 insert\_ulonglong() 723  
 insert\_ushort() 724  
 insert\_val() 724  
 insert\_wchar() 725  
 insert\_wstring() 726  
 IntegerTooLarge exception 795  
 Interceptor interface 1123  
 INTERDICTION\_POLICY\_ID 901  
 InterdictionPolicyValue 902  
 InterfaceDef Interface 139, 143  
 InterfaceDescription Structure 11  
 InterfaceRepository Quick Reference 2  
 InterfaceTypeMismatch exception 536, 557  
 INTERNAL exception 1280  
 INTF\_REPOS exception 1280  
 Introduction 1

- InvalidAddress exception 338
- InvalidConstraint exception 461
- InvalidControl exception 566
- InvalidEventType exception 425
- InvalidGrammar exception 460
- InvalidLookupRef exception 493
- InvalidName exception 329, 1136
- InvalidObjectRef exception 536
- InvalidPolicies exception 13
- InvalidPolicyValue 520
- InvalidSlot exception 1106
- INVALID\_TRANSACTION exception 567, 1281
- InvalidValue User Exception 726
- INV\_FLAG exception 1280
- INV\_IDENT exception 1281
- INV\_OBJREF exception 1281
- InvocationCredentialsPolicy 1261
- INVOCATION\_POLICIES constant 1080
- InvocationPolicyValue data type 571
- invoke() 228, 1175
- INV\_POLICY exception 1281
- IORInfo interface 1125
- IORInterceptor interface 1129
- IObject Interface 149
- is\_a() 167
- is\_a() 148, 279
- is\_abstract Attribute 280
- is\_ancestor\_transaction() 582
- is\_custom Attribute 280
- is\_descendant\_transaction() 583
- is\_equivalent() 167
- is\_nil() 8
- is\_related\_transaction() 583
- is\_same\_transaction() 584
- is\_system\_exception() 1087
- is\_top\_level\_transaction() 584
- Istring 488
- Istring data type 319
- IT\_Certificate
  - AVA interface 787
  - AVAList interface 791
  - Extension interface 797
  - ExtensionList interface 799
  - IT\_X509CertFactory interface 805
  - X509Certificate interface 795, 803
- IT\_Config module 809
- IT\_CosTransactions module 841
- it\_demarshal\_value() 1087
- item() 86, 136, 159
- Iterator::destroy() 658
- Iterator::get() 657
- IT\_EventChannelAdmin::ChannelAlreadyExists 896
- IT\_EventChannelAdmin::ChannelID Type 895
- IT\_EventChannelAdmin::ChannelNotFound 896
- IT\_EventChannelAdmin::EventChannelFactory
  - Interface 897
- IT\_EventChannelAdmin::EventChannelInfoList
  - Sequence 895
- IT\_EventChannelAdmin::EventChannelInfo
  - Structure 895
- IT\_EventChannelAdmin Module 895
- IT\_FPS::InterdictionPolicy Interface 903
- IT\_FPS Module 901
- it\_get\_fw\_type\_id() 1087
- it\_get\_safe\_bases() 1088
- IT\_LOG\_MESSAGE() macro 935
- IT\_LOG\_MESSAGE\_1() macro 936
- IT\_LOG\_MESSAGE\_2() macro 936
- IT\_LOG\_MESSAGE\_3() macro 936
- IT\_LOG\_MESSAGE\_4() macro 937
- IT\_LOG\_MESSAGE\_5() macro 938
- it\_marshall\_value() 1088
- IT\_NamingContextExt Interface 965
- IT\_NotifyLogAdmin 991
- IT\_NotifyLogAdmin::NotifyLog 993
- IT\_NotifyLogAdmin::NotifyLog::obtain\_offered\_types
  - () 993
- IT\_NotifyLogAdmin::NotifyLog::obtain\_subscribed\_types
  - () 993
- IT\_NotifyLogAdmin::NotifyLogFactory 995
- IT\_NotifyLogAdmin::NotifyLogFactory::default\_filter
  - factory 995
- IT\_NotifyLogAdmin::NotifyLogFactory::manager 995
- IT\_PortableServer::DISPATCH\_WORKQUEUE\_POLICY\_ID 1003
- IT\_PortableServer::DispatchWorkQueuePolicy
  - Interface 1005
- IT\_PortableServer module 1001
- it\_type() 1088
- IT\_TypedEventChannelAdmin::TypedEventChannelFactory
  - Interface 1043
- IT\_TypedEventChannelAdmin::TypedEventChannelInfoList
  - Sequence 1041
- IT\_TypedEventChannelAdmin::TypedEventChannelInfo
  - Structure 1041
- IT\_TypedEventChannelAdmin Module 1041
- IT\_WorkQueue 1047

IT\_X509CertFactory interface 805

## K

kind() 254

kind Attribute 215

## L

length() 255

length Attribute 42

LifespanPolicy class 1183

LIFESPAN\_POLICY\_ID constant 1164

LifespanPolicyValue enumeration 1164

lifetime filter attribute 364, 394

LimitingFollowTooPermissive exception 511

link if attribute 545

LinkInfo 510

LinkName 488

LinkNameSeq 488

list() 330

list\_changed() 824

Listener::variable\_added() 822

Listener::variable\_removed() 822

Listener interface 821

ListenerTargetRange enumeration 810

list\_initial\_services() 195

list\_links() 513

list\_offers() 498

ListOption 553

list\_proxies() 499

list\_types() 560

\_local\_narrow() 1088, 1091, 1094, 1095, 1097

Log::copy() 672

Log::copy\_with\_id() 672

Log::delete\_records() 668

Log::delete\_records\_by\_id() 669

Log::flush() 672

Log::get\_administrative\_state() 663

Log::get\_availability\_status() 665

Log::get\_capacity\_alarm\_thresholds() 665

Log::get\_current\_size() 662

Log::get\_forwarding\_state() 663

Log::get\_interval() 664

Log::get\_log\_full\_action() 662

Log::get\_log\_qos() 666

Log::get\_max\_record\_life() 661

Log::get\_max\_size() 662

Log::get\_n\_records() 662

Log::get\_operational\_state() 664

Log::get\_record\_attribute() 671

Log::get\_week\_mask() 666

Log::id() 661

Log::match() 668

Log::my\_factory() 661

Log::query() 667

Log::retrieve() 667

Log::set\_administrative\_state() 663

Log::set\_capacity\_alarm\_thresholds() 665

Log::set\_forwarding\_state() 664

Log::set\_interval() 664

Log::set\_log\_full\_action() 663

Log::set\_log\_qos() 666

Log::set\_max\_record\_life() 661

Log::set\_max\_size() 662

Log::set\_record\_attribute() 670

Log::set\_records\_attribute() 671

Log::set\_week\_mask() 666

Log::write\_recordlist() 670

Log::write\_records() 669

LOG\_ALL\_EVENTS 933

LOG\_ALL\_INFO 933

LOG\_ERROR 933

LOG\_FATAL\_ERROR 933

LOG\_INFO 933

LOG\_INFO\_HIGH 933

LOG\_INFO\_LOW 933

LOG\_INFO\_MED 933

LogMgr::find\_log() 675

LogMgr::list\_logs() 675

LogMgr::list\_logs\_by\_id() 675

LOG\_NO\_EVENTS 933

LogStream Interface 945

LOG\_WARNING 933

long\_changed() 824

lookup() 77

lookup\_id() 223

lookup\_if attribute 545

lookup\_name() 77

lookup\_value\_factory() 196

## M

make\_domain\_manager() 51

Manager interface 953, 955

MandatoryProperty exception 536

ManualWorkQueue 1053

dequeue() 1053

do\_work() 1053

shutdown() 1054

ManualWorkQueueFactory  
   create\_work\_queue() 1055  
 MappingFilter interface 477  
 marshal() 90  
 marshaled\_exception() 1088  
 marshaled\_exception\_seq sequence 1088  
 MARSHAL exception 1281  
 mask\_type() 560  
 match() 468, 483  
 match\_structured() 468, 484  
 MaxChainLengthPolicy 1027  
 max\_follow\_policy attribute 506  
 max\_hop\_count attribute 506  
 max\_left() 525, 527  
 max\_link\_follow\_policy 515  
 max\_list attribute 506  
 max\_match\_card attribute 506  
 max\_return\_card attribute 507  
 max\_search\_card attribute 507  
 Mechanism Policy 1263  
 member() 759  
 member\_count() 255  
 Member data type 918  
 MemberId data type 917  
 MemberIdList data type 918  
 member\_kind() 759  
 member\_label() 255  
 member\_name() 256, 760  
 members() method 924  
 members Attribute 127, 133, 247, 264  
 member\_type() 257  
 member\_visibility() 257  
 Messaging 1079  
 MissingMandatoryProperty exception 494  
 mode Attribute 43, 172  
 modify\_constraints() 466  
 modify\_link() 513  
 modify\_mapping\_constraints() 480  
 ModuleDef Interface 151  
 ModuleDescription Structure 13  
 move() 56  
 MULTIPLE\_ID 1164  
 MyChannel attribute 363  
 MyID attribute 363, 419  
 MyOperator attribute 364

## N

name() 153, 258  
 name Attribute 57

name attribute 1123  
 NameComponent structure 319  
 NamedValue Class 153  
 NameDynAnyPair Structure 692  
 Name sequence 319  
 NameValuePair Structure 693  
 NamingContextExt interface 337  
 NamingContext interface 323  
 NativeDef Interface 155  
 nested transactions 601  
 new\_context() 330  
 new\_for\_consumers() 372  
 new\_for\_suppliers() 372  
 next() 727  
 next\_n() 321, 525, 527  
 next\_one() 322  
   \_nil() 8, 197  
 NoContext Exception 1174  
 NO\_IMPLEMENT exception 1282  
 NO\_IMPLICIT\_ACTIVATION 1164  
 NoMatchingOffers exception 536  
 NO\_MEMORY exception 1282  
   \_non\_existent() 168  
 NON\_RETAIN 1167  
 NonTxTargetPolicyValue data type 572  
 NO\_PERMISSION exception 1282  
 NO\_RESOURCES exception 1282  
 NO\_RESPONSE exception 1282  
 NoSuchGroup exception 920  
 NoSuchMember exception 919  
 NotConnected exception 359  
 NotEmpty exception 331  
 NotFound exception 331  
 NotFoundReason enumeration 332  
 NotifyLog::create() 685  
 NotifyLog::get\_filter() 683  
 NotifyLog::set\_filter() 683  
 NotifyPublish interface 427  
 NotifySubscribe interface 429  
 NotImplemented exception 494  
 NotMasked exception 557  
 NotPrepared exception 566  
 NotProxyOfferId exception 531  
 NoTransaction exception 566  
 NotSubtransaction exception 566  
 NVList Class 157

## O

OBJ\_ADAPTER exception 1282

- Object Class 161
  - ObjectDeactivationPolicy class 1007
  - OBJECT\_DEACTIVATION\_POLICY\_ID constant 1001
  - ObjectDeactivationPolicyValue enumeration 1001
  - ObjectGroupFactory Interface 927
  - ObjectGroup Interface 921
  - ObjectId 197
  - object\_id Attribute 1153
  - ObjectIdList Sequence Class 197
  - ObjectId sequence 1165
  - ObjectId\_to\_string() 1162
  - ObjectId\_to\_wstring() 1162
  - ObjectId type 1137
  - OBJECT\_NOT\_EXIST exception 1283
  - object\_to\_string() 197
  - obtain\_notification\_pull\_consumer() 421
  - obtain\_notification\_pull\_supplier() 365
  - obtain\_notification\_push\_consumer() 421
  - obtain\_notification\_push\_supplier() 366
  - obtain\_offered\_types() 395
  - obtain\_subscription\_types() 380
  - Offer 491
  - offer\_change() 427, 983
  - OfferId 488
  - OfferIdIterator 525
  - OfferIdSeq 488
  - OfferInfo structure 535
  - OfferIterator 527
    - destroy() 527
  - OfferSeq 488
  - one-phase commit 599
  - operation() 228, 236
  - operation attribute 1147
  - operation\_context attribute 1147
  - OperationDef Interface 171
  - OperationDescription Structure 15
  - OperationMode Enumeration 16
  - operator=() Assignment Operators 241, 285
  - op\_name() 236
  - ORB Class 175
  - ORB\_CTRL\_MODEL 1168
  - orb\_id attribute 1137
  - ORBid Type 16
  - ORB\_init() 8
  - ORBInitializer interface 1131
  - ORBInitInfo interface 1133
  - OrbixEventsAdmin::ChannelManager 1099, 1101
  - Ordering type 1080
  - original\_type\_def Attribute 30, 269
  - OTSPolicyValue, Orbix 2000 enhancements 841
  - OTSPolicyValue data type 570
  - out() 242, 286
  - own\_credentials 1275
- ## P
- ParameterDescription Structure 16
  - ParameterList 690
  - ParameterMode Enumeration 17
  - Parameter structure 689
  - params() 237
  - params Attribute 172
  - parent() 82
  - perform\_work() 198
  - PERMIT 573
  - PersistenceModePolicy class 1009
  - PERSISTENCE\_MODE\_POLICY\_ID constant 1002
  - PersistenceModePolicyValue enumeration 1002
  - PERSISTENT 1165
  - PERSIST\_STORE exception 1283
  - pick() method 922
  - POA Class 1185
  - POAManager class 1209
  - Policy 492
  - PolicyCurrent class 211
  - PolicyErrorCode Type 17
  - PolicyError Exception 17
  - PolicyFactory interface 1141
  - Policy Interface 205
  - PolicyList Sequence 18
  - PolicyManager class 213
  - PolicyName 489
  - PolicyNameSeq 489
  - PolicySeq 489
  - policy\_type Attribute 208
  - PolicyTypeMismatch 520
  - PolicyType Type 19
  - PolicyValue 489
  - PolicyValueSeq sequence 1080
  - PolicyValue structure 1080
  - poll\_next\_response() 198
  - poll\_response() 229
  - PortableInterceptor module 1105
  - PortableServer module 1161
  - post\_init() 1131
  - postinvoke() 1224
  - PrecisionLogStream Interface 949

preface xxvii  
 Preference 518  
 pre\_init() 1132  
 preinvoke() 1224  
 prepare() 598  
 PREVENT 573  
 \_primary\_interface() 1175  
 PrimitiveDef Interface 215  
 PrimitiveKind Enumeration 19  
 PrincipalAuthenticator  
   authenticate() 1266  
 principal\_authenticator 1274  
 priority\_filter 394  
 priority\_filter attribute 364  
 PriorityRange structure 1080  
 Priority Type 1080  
 PropertNameSeq 489  
 Property 492  
 PropertyMode 554  
 PropertyName 489  
 PropertySeq 489  
 PropertyTypeMismatch exception 494  
 PropertyValue 490  
 PropStruct 555  
 PropStructSeq 553  
 Proxy 529  
 ProxyConsumer interface 379  
 proxy\_if attribute 545  
 ProxyInfo 530  
 ProxyNotFound exception 359  
 ProxyOfferId exception 537  
 ProxyPullConsumer interface 297, 383  
 ProxyPullSupplier interface 299, 385  
 ProxyPushConsumer interface 387  
 ProxyPushSupplier interface 303, 389  
 ProxySupplier interface 393  
 PullConsumer::disconnect\_pull\_consumer() 309  
 PullConsumer interface 309  
 pull\_structured\_event() 451  
 PullSupplier interface 311  
 pull\_suppliers attribute 365  
 push() 985  
 PushConsumer::disconnect\_push\_consumer() 313  
 PushConsumer::push() 313  
 PushConsumer interface 313  
 push\_structured\_event() 453, 989  
 push\_structured\_events() 445, 987  
 PushSupplier::disconnect\_push\_supplier() 315  
 PushSupplier interface 315

push\_suppliers attribute 365

## Q

QOPPolicy 1269  
 QoSAdmin  
   get\_qos() 351  
   set\_qos() 351  
   validate\_qos() 352  
 query() 520

## R

random\_groups() method 930  
 read\_Abstract() 95  
 read\_any() 95  
 read\_any\_array() 95  
 read\_boolean() 96  
 read\_boolean\_array() 96  
 read\_char() 97  
 read\_char\_array() 97  
 read\_double() 98  
 read\_double\_array() 98  
 read\_float() 98  
 read\_float\_array() 99  
 read\_long() 99  
 read\_long\_array() 99  
 read\_longdouble() 100  
 read\_longlong\_array() 100  
 read\_Object() 101  
 read\_octet() 101  
 read\_octet\_array() 101  
 ReadonlyDynamicProperty exception 494  
 ReadonlyProperty exception 537  
 read\_short() 102  
 read\_short\_array() 102  
 read\_string() 103  
 read\_TypeCode() 103  
 read\_ulong() 103  
 read\_ulong\_array() 103  
 read\_ulonglong() 104  
 read\_ulonglong\_array() 104  
 read\_ushort() 105  
 read\_ushort\_array() 105  
 read\_Value() 106  
 read\_wchar() 106  
 read\_wchar\_array() 106  
 read\_wstring() 107  
 rebind() 332

## INDEX

rebind\_context() 333  
REBIND exception 1283  
rebind\_mode() 1092  
RebindMode type 1080  
~RebindPolicy() 1092  
RebindPolicy Class 1091  
received\_credentials 1247, 1257, 1273, 1274, 1275  
received\_exception attribute 1112  
received\_exception\_id attribute 1113  
receive\_exception() 1116  
receive\_other() 1117  
receive\_reply() 1118  
receive\_request() 1156  
receive\_request\_service\_contexts() 1156  
RecoveryCoordinator class 595  
recreate() 610  
RefCountedLocalObject() constructor 829  
RefCountedLocalObject class 829  
RefCountedLocalObjectNC() constructor 831  
RefCountedLocalObjectNC class 831  
\_refcount\_value() 266  
reference\_to\_id() 1203  
reference\_to\_servant() 1204  
Register  
    modify() 540  
register\_if attribute 546  
register\_initial\_reference() 1137  
Register interface 535  
RegisterNotSupported exception 537  
register\_policy\_factory() 1138  
register\_resource() 585  
register\_stream() 941  
register\_subtran\_aware() 585  
register\_synchronization() 586  
register\_value\_factory() 198  
related documentation xviii  
\_release() 168  
remove() 86, 136, 159  
remove\_all\_constraints() 467  
remove\_all\_filters() 472  
remove\_all\_mapping\_constraints() 483  
remove\_filter() 471  
remove\_link() 514  
remove\_listener() 819  
remove\_member() method 923  
remove\_own\_credentials() 1274  
\_remove\_ref() 266, 281, 829, 832  
remove\_ref() 8  
remove\_type() 561  
replay\_completion() 595  
~ReplyHandler() 1094  
ReplyHandler Base class 1093  
reply\_status attribute 1147  
ReplyStatus type 1106  
report\_event() 942, 945, 950  
report\_message() 943, 946, 950  
RepositoryIdSeq Sequence 20  
RepositoryId Type 20  
Repository Interface 217  
Request Class 225  
RequestContext 690  
request\_id attribute 1148  
request\_id stem attribute 498  
RequestInfo interface 1143  
RequestProcessingPolicy class 1215  
REQUEST\_PROCESSING\_POLICY\_ID constant 1165  
RequestProcessingPolicyValue enumeration 1165  
RequestSeq Sequence 199  
REQUIRES 570  
resolve() 334  
resolve\_initial\_references() 199, 1139  
resolve\_str() 338  
response\_expected attribute 1148  
result() 229, 238  
result Attribute 173  
result attribute 1149  
result\_def Attribute 173  
resume() 592  
resume\_connection() 405  
RETAIN 1167  
\_retn() 243, 287  
return\_value() 229  
rewind() 727  
rollback() 593, 599, 606  
rollback\_only() 587, 593  
rollback\_subtransaction() 602  
~RoutingPolicy() 1095  
RoutingPolicy class 1095  
routing\_range() 1096  
RoutingTypeRange structure 1082  
RoutingType type 1081  
rr\_groups() method 930  
run() 200



**S**

- scale Attribute 137
- scope, configuration 809
- SecurityLevel2
  - Current interface 1247, 1257
- SecurityManager
  - get\_target\_credentials() 1273, 1274, 1275
  - own\_credentials 1275
  - principal\_authenticator 1274
  - remove\_own\_credentials() 1274
- seek() 727
- SelectionMethod data type 918
- send\_deferred() 229
- send\_exception() 1157
- sending\_exception attribute 1153
- send\_multiple\_requests\_deferred() 201
- send\_multiple\_requests\_oneway() 201
- send\_oneway() 230
- send\_other() 1158
- send\_poll() 1118
- send\_reply() 1159
- send\_request() 1119
- SequenceDef Interface 233
- SequenceProxyPullConsumer interface 397
- SequenceProxyPullSupplier interface 401
- SequenceProxyPushConsumer interface 399
- SequenceProxyPushSupplier interface 403
- SequencePullConsumer interface 439
- SequencePullSupplier interface 441
- SequencePushConsumer interface 445
- SequencePushSupplier interface 447
- Sequences 5
- ServantActivator class 1217
- ServantLocator Class 1223
- ServantManager Interface 1227
- Servant native type 1166
- ServantRetentionPolicy class 1229
- SERVANT\_RETENTION\_POLICY\_ID constant 1167
- ServantRetentionPolicyValue enumeration 1167
- servant\_to\_id() 1204
- servant\_to\_reference() 1205
- ServerRequest Class 235
- ServerRequestInfo interface 1151
- ServerRequestInterceptor interface 1155
- SERVER\_SIDE 841
- ServiceTypeExists exception 557
- ServiceTypeName 490
- ServiceTypeNameSeq 553
- ServiceTypeRepository Interface 553
- SessionCachingPolicy 1029
- set\_as\_string() 740
- set\_as\_ulong() 741
- set\_def\_follow\_policy() 499
- set\_def\_hop\_count() 500
- set\_def\_match\_card() 500
- set\_def\_return\_card() 500
- set\_def\_search\_card() 501
- set\_discriminator() 760
- set\_elements() 736, 747
- set\_elements\_as\_dyn\_any() 737, 747
- set\_exception() 236
- set\_filter() 943
- set\_length() 748
- set\_max\_follow\_policy() 501
- set\_max\_hop\_count() 501
- set\_max\_link\_follow\_policy() 501
- set\_max\_list() 502
- set\_max\_match\_card() 502
- set\_max\_return\_card() 502
- set\_max\_search\_card() 503
- set\_members() 753, 766
- set\_members\_as\_dyn\_any() 754, 767
- set\_member\_timeout() method 925
- set\_one\_value() 82
- SetOverrideType Enumeration 21
- set\_policy\_overrides() 214
- set\_request\_id\_stem() 503
- set\_result() 239
- set\_return\_type() 230
- set\_servant() 1206
- set\_servant\_manager() 1207
- set\_slot() 1122, 1153
- set\_supports\_dynamic\_properties() 503
- set\_supports\_modifiable\_properties() 503
- set\_supports\_proxy\_offers() 504
- set\_timeout() 594
- set\_to\_default\_member() 761
- set\_to\_no\_active\_member() 761
- set\_type\_repos() 504
- set\_value() 744
- set\_values() 83
- SHARED 571
- shutdown() 820
- SINGLE\_THREAD\_MODEL 1168
- SlotId type 1106
- SpecifiedProps 519
- SpecifiedServiceTypes 556
- State enumeration 1213

StatusActive 568  
 StatusCommitted 568  
 StatusCommitting 569  
 Status enumeration type 568  
 StatusMarkedRollback 568  
 StatusNoTransaction 569  
 StatusPrepared 568  
 StatusPreparing 569  
 StatusRolledBack 569  
 StatusRollingBack 569  
 StatusUnknown 569  
 string\_alloc() 8  
 string\_changed() 825  
 StringDef Interface 245  
 string\_dup() 8  
 StringName data type 339  
 string\_to\_object() 203  
 string\_to\_ObjectId() 1162  
 String\_var() Constructors 242  
 ~String\_var() Destructor 243  
 String\_var Class 241  
 StructDef Interface 247  
 StructMember Structure 21  
 StructuredProxyPullConsumer interface 407  
 StructuredProxyPullSupplier interface 409  
 StructuredProxyPushConsumer interface 411  
 StructuredProxyPushSupplier interface 413  
 StructuredPullConsumer interface 449  
 StructuredPullSupplier interface 451  
 StructuredPushConsumer interface 453  
 StructuredPushSupplier interface 455  
 subscription\_change() 429  
 SubsystemId data type 934  
 SubtransactionAwareResource class 601  
 SubtransactionsUnavailable exception 566  
 SupplierAdmin::obtain\_pull\_consumer() 305  
 SupplierAdmin::obtain\_push\_consumer() 305  
 SupplierAdmin interface 417  
 SupportAttributes interface 543  
 supported\_interfaces Attribute 280  
 supports\_dynamic\_properties attribute 543  
 supports\_modifiable\_properties attribute 543  
 supports\_proxy\_offers attribute 543  
 suspend() 594  
 suspend\_connection() 391, 405, 415  
 synchronization() 1098  
 Synchronization class 603  
 sync\_scope attribute 1149  
 ~SyncScopePolicy() 1098

SyncScopePolicy class 1097  
 SyncScope type 1082  
 system exceptions 1279  
 SYSTEM\_ID 1163

## T

TAG\_POLICIES constant 1083  
 target() 231  
 target attribute 1113  
 target\_is\_a() 1154  
 target\_most\_derived\_interface attribute 1154  
 TCKind Enumeration 21  
 Terminator class 605  
 the\_activator() 1208  
 The DynamicAny Module 691  
 The IT\_CORBA Module 827  
 The IT\_LoadBalancing module 917  
 The IT\_Logging module 931  
 The IT\_Naming module 963  
 the\_name() 1207  
 the\_parent() 1207  
 the\_POAManager() 1207  
 \_this() 1175  
 ThreadPolicy class 1231  
 THREAD\_POLICY\_ID constant 1167  
 ThreadPolicyValue enumeration 1167  
 threads\_total Attribute 1049  
 Timestamp data type 934  
 to\_any() 728  
 to\_name() 339  
 to\_string() 339  
 to\_url() 340  
 TraderComponents 545  
 TraderName 490  
 TransactionalObject class 607  
 TransactionFactory class 609  
 TransactionPolicyValue data type 573  
 TRANSACTION\_REQUIRED exception 567, 1283  
 TRANSACTION\_ROLLEDBACK exception 567,  
 1283  
 TRANSIENT 1165  
 TRANSIENT exception 1284  
 TrustedCAGroupPolicy 1033, 1035, 1037, 1039  
 try\_pull\_structured\_events() 442  
 two-phase commit 598  
 type() 728  
 type Attribute 44, 48, 134, 142, 234, 245, 269,  
 283, 289  
 TypeCode Class 249

TypedConsumerAdmin::obtain\_typed\_push\_supplier() 614  
 TypedConsumerAdmin::obtain\_typed\_pull\_supplier() 613  
 type\_def Attribute 44, 48, 283  
 TypedefDef Interface 261  
 TypeDescription Structure 22  
 TypedEventChannelFactory::create\_typed\_channel() 1043  
 TypedEventChannelFactory::find\_typed\_channel() 1044  
 TypedEventChannelFactory::find\_typed\_channel\_by\_id() 1044  
 TypedEventChannelFactory::list\_typed\_channels() 1044  
 TypedPushConsumer::get\_typed\_consumer() 623  
 TypedSupplierAdmin::obtain\_typed\_pull\_consumer() 620  
 TypedSupplierAdmin::obtain\_typed\_push\_consumer() 619  
 TypeMismatch exception 820  
 TypeMismatch User Exception 729  
 type\_modifier() 259  
 type\_repos attribute 543  
 TypeRepository 490  
 TypeStruct 555  
 typographic conventions xviii

## U

Unavailable exception 567  
 unbind() 335  
 UnionDef Interface 263  
 UnionMember Structure 23  
 UNIQUE\_ID 1163  
 unknown\_adapter() 1169  
 UNKNOWN exception 1284  
 UnknownLinkName exception 511  
 UnknownMaxLeft exception 495  
 UnknownOfferId exception 495  
 UnknownPropertyName exception 537  
 UnknownServiceType exception 495  
 UnknownTraderName exception 538  
 unmarshal() 90  
 unmask\_type() 562  
 unregister\_value\_factory() 203  
 UNSHARED 571  
 UnsupportedFilterableData exception 460  
 update\_member\_load() method 924  
 URLString data type 340

USE\_ACTIVE\_OBJECT\_MAP\_ONLY 1166  
 USE\_DEFAULT\_SERVANT 1166  
 USER\_ID 1163  
 USE\_SERVANT\_MANAGER 1166

## V

validate\_event\_qos() 380, 395  
 value() 1154, 1007, 1009, 1177, 1179, 1181, 1183, 1216, 1229, 1231  
 value Attribute 48  
 ValueBase() Constructors 266  
 ~ValueBase() Destructor 266  
 ValueBase Class 265  
 ValueBoxDef Interface 269  
 ValueDef Interface 271  
 ValueDescription Structure 23  
 ValueFactory 281  
 ValueFactoryBase() Constructor 281  
 ~ValueFactoryBase() Destructor 281  
 ValueFactoryBase Class 281  
 ValueFactory Type 281  
 ValueMemberDef Interface 283  
 ValueMember Structure 24  
 value\_type 479  
 Value\_Type Quick Reference 4  
 ValueTypeRedefinition exception 557  
 version Attribute 57  
 VoteCommit 569, 598  
 Vote enumeration type 569  
 VoteReadOnly 569, 598  
 VoteRollback 569, 598

## W

WellKnownAddressingPolicy class 833  
 WELL\_KNOWN\_ADDRESSING\_POLICY\_ID Constant 827  
 withdraw() 541  
 withdraw\_proxy() 533  
 withdraw\_using\_constraint() 542  
 WorkItem 1057  
   Destroy 1057  
   execute() 1057  
 work\_pending() 203  
 WorkQueue 1059  
   activate() 1060  
   deactivate() 1061  
   enqueue() 1059  
   flush() 1061

## INDEX

- is\_empty() 1060
- is\_full() 1060
- WorkQueue::enqueue\_immediate() 1060
- WorkQueue::owns\_current\_thread() 1061
- WorkQueuePolicy 1063
- write\_any() 111
- write\_any\_array() 111
- write\_boolean() 112
- write\_boolean\_array() 112
- write\_char() 113
- write\_char\_array() 113
- write\_double() 113
- write\_double\_array() 114
- write\_float() 114
- write\_float\_array() 115
- write\_long() 115
- write\_long\_array() 116
- write\_longlong() 116
- write\_longlong\_array() 116
- write\_Object() 117
- write\_octet() 117
- write\_octet\_array() 118
- write\_short() 118
- write\_short\_array() 119
- write\_string() 119
- write\_TypeCode() 119
- write\_ulong() 120
- write\_ulong\_array() 120
- write\_ulonglong() 121
- write\_ulonglong\_array() 121
- write\_ushort() 122
- write\_ushort\_array() 122
- write\_Value() 122
- write\_wchar() 123
- write\_wchar\_array() 123
- write\_wstring() 124
- WstringDef Interface 289
- wstring\_to\_ObjectId() 1162
- WString\_var() Constructors 286
- ~WString\_var() Destructor 287
- WString\_var Class 285

## X

- X509Certificate interface
  - get\_der\_serial\_number() 806
  - IntegerTooLarge exception 795







