

Appendix J

CORBA Compliance

J.1 Introduction

This appendix contains detailed information about TAO's compliance with various OMG CORBA specifications. The information in this appendix should not be considered as a legally binding statement. It reflects the intent of the developers of TAO, based on their interpretation of the various OMG specifications. There are presently no detailed, generally recognized acceptance tests at the level of the OMG specifications listed in this appendix.

Whereas compliance with OMG specifications is a design goal of TAO, it is also a continuing pursuit. As a user of TAO, you can help in this pursuit by contacting us if you feel TAO's implementation misinterprets any part of the cited OMG specifications, or even better, by contributing ideas and source code for how TAO's compliance could be improved.

Note *Throughout this document, "TAO" and "TAO 1.1a" refer to "OCI's Distribution of TAO, Version 1.1a."*

TAO is intended to be compliant with the following specifications:

- CORBA/IIOP 2.3.1
- CORBA C++ language mapping
- CORBA Interoperable Naming Service
- CORBA Messaging
- Real-Time CORBA 1.0

It is also possible to build TAO such that it is compliant with the Minimum CORBA specification. The following sections list specific features of each of these specifications that are supported, and also describe where TAO deviates from each specification.

J.2 CORBA/IIOP 2.3.1

TAO is intended to be compliant with the OMG CORBA/IIOP 2.3.1 specification (formal/99-10-07). The CORBA specification defines three separate compliance points:

- CORBA Core
- CORBA Interoperability
- CORBA Interworking

The minimum required for a CORBA-compliant system is adherence to the specifications in CORBA Core and one language mapping. See J.3 for information about TAO's C++ language mapping compliance. We address TAO's compliance with CORBA Core, CORBA Interoperability, and CORBA Interworking below.

J.2.1 CORBA Core

TAO supports the CORBA Core with the following exceptions:

- Abstract interfaces are not supported.
- Valuetypes are only partially supported, as needed to support the generation of exception holder classes for the Asynchronous Method Invocation callback model portion of the CORBA Messaging specification (see J.5).

- Fixed data types are not supported.
- Domain Managers are not supported.
- The Interface Repository is not yet fully implemented.
- DynAny support is based on CORBA 2.2, not the updated DynAny specification in CORBA 2.3.

J.2.2 CORBA Interoperability

An ORB is considered to be interoperability-compliant when it meets the following requirements:

- In the CORBA Core part of the specification, standard APIs are provided by an ORB to enable the construction of request-level inter-ORB bridges. APIs are defined by the Dynamic Invocation Interface (DII), the Dynamic Skeleton Interface (DSI), and by the object identity operations described in the Interface Repository chapter of the specification.
- The Internet Inter-ORB Protocol (IIOP) defines a transfer syntax and message formats (described independently as the General Inter-ORB Protocol), and defines how to transfer messages via TCP/IP connections. The IIOP can be supported natively or via a half-bridge.

TAO fully supports the DII and DSI APIs. TAO also fully supports the object identity operations, such as `CORBA::Object::_is_equivalent()` and `CORBA::Object::_is_a()`. TAO implements IIOP as a pluggable protocol, and is interoperable with ORBs that support IIOP version 1.0, 1.1, or 1.2.

Note *TAO's implementation of GIOP/IIOP 1.2 does not support bidirectional communications.*

J.2.3 CORBA Interworking

The purpose of the Interworking architecture is to specify support for two-way communication between CORBA objects and COM objects. The goal is for objects from one object model to be able to be viewed as if they existed in the other object model. For example, a client working in a CORBA model should be able to view a COM object as if it were a CORBA object. Likewise, a client working in a COM object model should be able to view a CORBA object as if it were a COM object.

TAO does not support the CORBA Interworking architecture specification.

J.3 C++ Language Mapping

TAO is intended to be compliant with the CORBA 2.3 C++ language mapping (formal/99-07-41), except for the elements of the CORBA Core specification that are not supported as described in J.2.1 (e.g., abstract interfaces, fixed data types, valuetypes).

In addition, TAO supports alternative mappings for modules and exceptions, as described in the CORBA 2.3 C++ language mapping specification.

J.3.1 Modules

In addition to the standard mapping for modules, TAO supports the alternative mappings for C++ dialects that do not support the namespace construct, as defined in Section 1.42.1 of formal/99-07-41.

J.3.2 Exceptions

In addition to the standard mapping for exceptions, TAO supports the alternative mapping for C++ dialects that do not support real C++ exceptions, as defined in Section 1.42.2 of formal/99-07-41. See Chapter 8 for more information about exceptions and error handling in TAO.

J.4 Interoperable Naming Service

TAO implements the Interoperable Naming Service specification as defined in OMG TC document orbos/98-10-11, with the following exceptions:

- The `iiopname` URL syntax for object references is not supported.
- The `NamingContextExt` interface is not supported.

In addition to the `IOR` and `iioploc` formats, TAO supports the following URL schemes:

```
file
iiop
uiop or uioploc
shmiop or shmioploc
```

The `iiop`, `uiop`, `uioploc`, `shmiop`, and `shmioploc` schemes are based on pluggable protocol implementations that are supplied with TAO. Additional pluggable protocols, and their associated object URL schemes, are possible. See Chapter 13 for more information on pluggable protocols.

Note *The OMG has recently issued a revised Interoperable Naming Service specification (ptc/99-12-03) that changes the syntax of object URL schemes to use `corbaloc` and `corbaname` prefixes and accommodates multiple protocols. TAO 1.1a does not support these new object URL schemes.*

J.5 CORBA Messaging

TAO implements a portion of the CORBA Messaging specification as defined in OMG TC document orbos/98-05-05. Support for CORBA Messaging is enabled by default when the TAO and `orbsvcs` libraries are built. The precompiler macros `TAO_HAS_CORBA_MESSAGING` and `TAO_HAS_AMI` are used to enable/disable the features of the CORBA Messaging specification and are defined in `$TAO_ROOT/tao/orbconf.h`. These features can also be enabled/disabled using the `corba_messaging` and `ami` GNU Make flags. See 2.3 for more information on choosing build flags.

TAO supports the following features of the CORBA Messaging specification:

- Asynchronous Method Invocation (AMI) (*callback* model only).

TAO differs from the OMG specification for AMI callback in the following way. The specification reads:

When the callback model is used, the client supplies a reply handler when making the asynchronous invocation. The interface's operations and attributes are mapped to implied-IDL operations with names prefixed by "`sendc_`". If this implied-IDL operation name conflicts with existing operations on the interface or any of the interface's base interfaces, "`ami_`" strings are inserted between "`sendc_`" and the original operation name until the implied-IDL operation name is unique.

Because using "`sendc_`" in two different contexts could become very confusing for the person maintaining the code, TAO's IDL compiler will instead reject the IDL file with an illegal redefinition error message if an implied-IDL operation name conflicts with existing operations on the interface or any of the interface's base interfaces.

- Quality of Service (QoS) Framework

Section 5.2 of orbos/98-05-05 describes a QoS framework. In this framework, all QoS settings are interfaces derived from `CORBA::Policy`. Client-side QoS policy management is performed through operations accessible in the context of an ORB, thread, or Object. Server-side policy management is performed via policies applied to a POA at creation.

- Messaging Quality of Service

Relative round-trip request invocation timeouts (`Messaging::RelativeRoundTripTimeoutPolicy`) are supported (for synchronous requests only).

Synchronization scope policy (`Messaging::SyncScopePolicy`) control for oneway operations is supported.

TAO does not support the following features of the CORBA Messaging specification:

- Asynchronous Method Invocation (AMI) *polling* model.
- Time-Independent Invocations (TII).
- Messaging QoS policies other than relative round-trip timeout and synchronization scope.

See Chapter 9 for more information about using the supported features of the CORBA Messaging specification with TAO.

J.6 Real-Time CORBA

TAO implements a portion of the Real-Time CORBA 1.0 specification as defined in OMG document ptc/99-06-02. Real-Time CORBA 1.0 is the first step taken by the OMG in providing comprehensive support for the needs of the real-time community. The Real-Time CORBA specification defines one mandatory compliance point, as defined in Chapter 4 of ptc/99-06-02. The Real-Time CORBA Scheduling Service, as described in Chapter 5 of ptc/99-06-02, is a separate and optional compliance point. TAO does not implement the Real-Time CORBA Scheduling Service.

Support for Real-Time CORBA is enabled by default when the TAO and orbsvcs libraries are built. The precompiler macro `TAO_HAS_RT_CORBA` is used to enable/disable Real-Time CORBA support and is defined in

`$TAO_ROOT/tao/orbconf.h`. Real-Time CORBA support can also be enabled/disabled using the `rt_corba` GNU Make flag. See 2.3 for more information on choosing build flags.

TAO supports the following features of the mandatory portion of the Real-Time CORBA specification:

- Interfaces defined in the RTCORBA module (`RTCORBA::RTORB`, `RTCORBA::Current`).
- CORBA priority mappings (both linear and direct).
- Explicit binding via `CORBA::Object::_validate_connection()`.
- Invocation timeouts via the `RelativeRoundtripTimeoutPolicy` as described in 9.3.1.

TAO does not support the following features of the mandatory portion of the Real-Time CORBA 1.0 specification:

- Server declared priorities.

TAO has a functionally equivalent *endpoint-per-priority* feature that allows you to associate a CORBA priority or priority range with each endpoint. These endpoint priorities are communicated to the client through the IOR. The `TAO::ClientPriorityPolicy`, a TAO-specific policy, can be used on the client side to allow the client to select the desired profile in the IOR, based on the priority of the calling thread or based on a priority range.

- Client-propagated priorities.
- Thread pools via the `ThreadPoolPolicy`.

TAO provides its own ORB-based thread pool model in which each thread in the pool implements its own ORB event loop, and in which requests are dispatched on the next available thread in the pool using the *leader/follower* pattern. For more information on the *leader/follower* pattern, see `$TAO_ROOT/docs/leader_follower.html` in your TAO source code distribution.

- Private connections.
- Protocol configuration.

TAO provides limited protocol configuration for TCP via TAO-specific ORB initialization options such as `Urbanology`, `ORBSndSock`, and `ORBRcvSock`. See Chapter 15 for more details.

- Priority transforms.
- Priority-banded connections.

Though TAO does not directly support priority-banded connections as defined in the RT CORBA specification, through the use of TAO's *endpoint-per-priority* feature and `ClientPriorityPolicy`, a functionally equivalent form of priority-banded connections is possible. Endpoints in an IOR may have different priorities associated with them, corresponding to a thread or pool of threads running at that priority on the server. Depending upon the value of the `ClientPriorityPolicy`, the client-side ORB will establish a connection to the appropriate endpoint, based on the priority of the calling thread. In a multithreaded client, this could result in multiple (*banded*) connections between the client and server ORBs (see Figure J-1). Such a configuration can help avoid priority inversion during request delivery and processing.

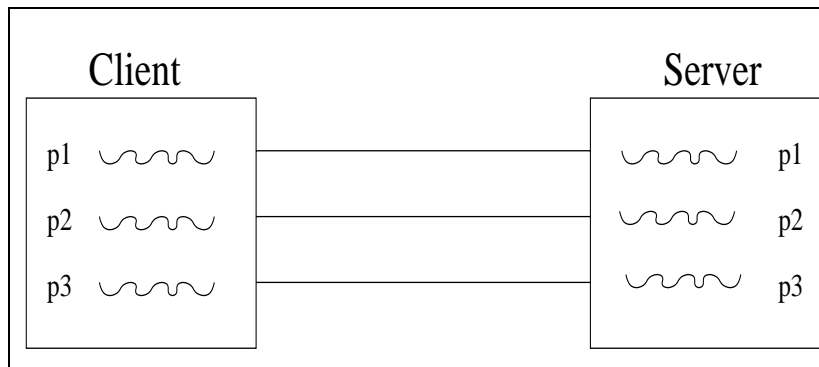


Figure J-1 Endpoint-per-Priority and Client Priority Policy

J.7 Minimum CORBA

TAO implements the *minimumCORBA* specification defined in OMG TC document orbos/98-08-04. This specification defines a profile (or subset) of CORBA that is designed for systems with limited resources. It attempts to satisfy the resource constraints of such systems while preserving portability,

interoperability, and full IDL support. Support for minimumCORBA is disabled by default when the TAO and orbsvcs libraries are built. The precompiler macro TAO_HAS_MINIMUM_CORBA is used to enable/disable minimumCORBA support and is defined in \$TAO_ROOT/tao/orbconf.h. Support can also be enabled/disabled using the minimum_corba GNU Make flag. See 2.3 for more information on choosing build flags.

Note *Enabling minimumCORBA support in TAO disables support for CORBA Messaging by default. CORBA Messaging can be enabled along with minimumCORBA using the macros or GNU Make flags described in J.5.*

The following features are disabled in TAO when minimumCORBA support is enabled:

- Dynamic Invocation Interface (DII).
- Dynamic Skeleton Interface (DSI).
- DynamicAny.
- Interface Repository.
- Implementation Repository.
- CORBA Messaging and AMI.
- Real-Time CORBA.
- Remote Policies.
- Interceptors.
- The following interfaces, exceptions, and types in module CORBA:
 - Context and ContextList
 - Request and RequestSeq
 - ServerRequest
 - ConstructionPolicy
 - NamedValue, NVList, and NameValuePair
 - WrongTransaction
 - AnySeq
 - DynAny, DynSequence, DynStruct, etc.
 - FieldName
 - ORB::InconsistentTypeCode
- The following operations on CORBA::Object:
 - _non_existent()
 - _get_implementation()

```
_get_interface()
_create_request()
_request()
```

- `PortableServer::ForwardRequest`.
- Support for UIOP pluggable protocol (on operating systems that support local IPC).
- Support for SHMIOP pluggable protocol.

In addition, the following advanced POA features in the `PortableServer` module are disabled by default when `minimumCORBA` support is enabled:

- The following CORBA policies:

```
ThreadPolicy
ImplicitActivationPolicy
ServantRetentionPolicy
RequestProcessingPolicy
```

- Servant Managers (`ServantActivator` and `ServantLocator`).
- Default Servant.
- Adapter Activators.
- The following POA Manager operations and associated POA Manager states:

```
hold_requests()
discard_requests()
deactivate()
```

- The following `UserException` types in interface `PortableServer::POA`:

```
AdapterInactive
NoServant
```

Note *The above advanced POA features can be enabled, even if `minimumCORBA` is enabled, by defining the precompiler macro `TAO_HAS_MINIMUM_POA` with a value of 0 in `$TAO_ROOT/tao/orbconf.h` before building TAO.*
