Curl CORBA Clients: Integrating Distributed Objects With the World Wide Web

by

Jason Sugg

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 21, 1997

Author .....
Department of Electrical Engineering and Computer Science
May 21, 1997

Certified by .....
Steve Ward
Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Curl CORBA Clients: Integrating Distributed Objects With the World Wide Web

by

Jason Sugg

Submitted to the
Department of Electrical Engineering and Computer Science
on May 21, 1997, in partial fulfillment of the requirements for the
degrees of
Master of Engineering in Electrical Engineering and Computer Science
and
Bachelor of Science in Computer Science and Engineering

Abstract

The object paradigm for distributed systems is rapidly replacing the traditional client-server model. This new paradigm creates sophisticated new ways of providing data and services to web-based clients. A partial implementation of the CORBA distributed object system in the Curl web programming language provides browser-based Curl clients with access to all CORBA objects accessible over the Internet. This implementation is built in such a way as to allow easy addition of new pathways to objects. This partial implementation leaves much room for future work building the system into a full CORBA ORB.

Thesis Supervisor: Steve Ward
Title: Professor

Table of Contents

Chapter 1

Introduction

1.1 Distributed Objects

For years, the sockets-based client-server model has been the dominant paradigm for distributed systems. The model fit well with the predominant computing environment — low-end client machines acting as user interfaces to larger, more powerful servers. But as computing power has become cheaper the network landscape has changed, and the traditional client-server approach has seized to fit as well as it used to. In light of these changes, the computing world has been witnessing a paradigm shift away from the traditional client-server model and towards a distributed object model in which the distinction between client and server is blurred, and the emphasis is instead on objects which may live anywhere on the network. Like their predecessor client-server systems, support for these distributed object systems will soon be necessary in all commercial programming systems.

In the old client-server model, a labor of coordination was required to build a working client and server. The two parties had to agree on what server port they were going to talk through, what functions were going to be provided by the server, and the manner of marshalling arguments between them. The introduction of the Remote Procedure Call (RPC) standard alleviated some of these worries by masking hardware differences and network layers, but left the machines and many of the worries that accompany them visible in the programming model.

Distributed object systems attempt to ease programmer headaches further by allowing programmers to see the network as a collection of objects providing and requesting services rather than as a collection of distinct machines running programs which wish to communicate with one another. This additional layer of abstraction adds a number of ben-

efits, including location transparency for objects, convenient directory services, a component software paradigm, and the ability to dynamically add and utilize new network services.

The exact meaning of the much-bantered term "object" depends greatly on which object system is being described, but both of the dominant distributed object systems provide data encapsulation mechanisms, and at least one of them also provides inheritance and polymorphism. The benefits these principles can offer to programming languages have long been recognized, but until recently it was not understood how to extend those benefits to systems as a whole.

1.2 Distributed Object Systems

Currently two distributed object systems dominate. The first is Microsoft's Component Object Model (COM). COM started as a lightweight, single-machine standard for encapsulating data and the services necessary to utilize that data for use between processes on the same machine, and has been extended in an ad hoc fashion to allow distributed sharing of services. The second system is the Common Object Request Broker Architecture (CORBA). CORBA is a heavy-duty object system providing both inheritance and polymorphism in addition to the encapsulation provided by COM. Defined as a standard by the Object Management Group (OMG), an industry consortium of over 500 companies, CORBA is rapidly becoming the object system of choice for commercial applications. Consequently, CORBA support greatly enhances the Curl programming system's potential as a future platform for building distributed applications.

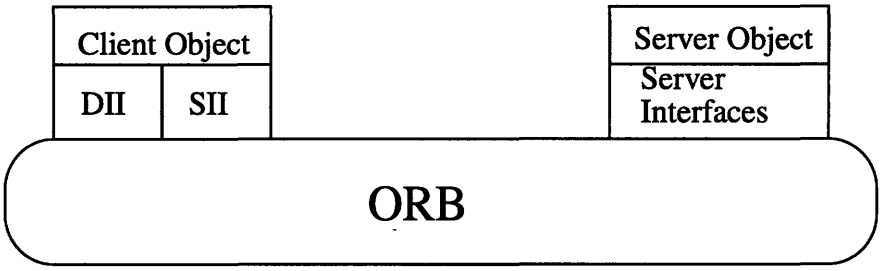

Figure 1.1: The Common Object Request Broker Architecture

1.3 CORBA

The layout of CORBA is shown in Figure 1. The terms "client" and "server" in the figure have meaning only in relation to specific transactions. In CORBA, instead of certain monolithic processes acting as servers all the time and other smaller processes as clients, each process is viewed as an object which can act as both a client and a server. Each object provides a specific small set of services, but may act as a client of other objects while performing those services. All communications between objects are brokered by an Object Request Broker (ORB). The ORB helps clients locate server objects, delivers requests and replies between client and server, and may provide other services such as verifying client and server identities for security purposes, providing generic access to persistent memory, or helping clients locate servers which fulfill a certain need.

1.4 Curl and CORBA

Web content could be greatly enhanced by making the CORBA object universe accessible to browser-based web programs. Variable data such as stock quotes or weather conditions could be fetched in a direct and convenient way from CORBA objects accessible via the Internet or secure web purchases made using CORBA's security service. In fact, it is possible that the web could eventually become fully integrated with the universe of distributed objects, so that a web browser might become an object browser which provides views of

objects and the services or content they provide. Likewise, the world of distributed objects could greatly benefit from the portability provided by browser-based clients.

But the problem of providing a browser-based client with access to CORBA objects presents some architectural and systemic problems. Most importantly, it is not obvious how to allow an arbitrary client running on an arbitrary browser to utilize the services of an arbitrary ORB. Since a web-based client could potentially be anywhere on the Internet, some mechanism must be devised for allowing the client access to an ORB's objects from outside that ORB's domain. Such a mechanism should preserve the security of the ORB, while providing access to all of the ORB's public services as well as its objects.

Fortunately, CORBA defines an ORB interoperability protocol which allows methods of an object in one ORB to be invoked from the domain of a different ORB while preserving security features. To utilize this protocol in Curl, a mini-ORB, or ORBlet was implemented which is downloaded to the browser along with the Curl program, and which knows how to communicate with other ORBs using the standard interoperability protocol. The ORBlet provides a standard client interface to the CORBA object universe for its browser-based clients, but relies entirely on other ORBs for its actual functionality. The ORBlet is not quite a full ORB because it does not currently support server interfaces. Consequently, browser-based objects may act only as clients of other objects, and not as servers.

The following chapters describe the interface and implementation of the Curl ORBlet. Chapter 2 describes the interface the ORBlet provides to client programmers. Chapter 3 describes the CORBA interoperability protocol which the ORBlet uses to communicate with server objects. Chapter 4 describes the ORBlet's implementation of that protocol, and Chapter 5 describes the type mappings from CORBA's Interface Definition Language to Curl.

Chapter 2

Request Generation and Invocation

2.1 CORBA Invocation Interfaces

CORBA was defined as a standard for allowing any object to share its services with other distributed objects without the restrictions normally imposed by network location, platform difference, or source language difference. To allow clients to utilize the services of other objects in this transparent fashion, CORBA defines two ORB client interfaces which differ in their generality and ease of use. The less general Static Invocation Interface (SII) is very similar to the RPC interface. As with RPC, specifications of the server's interfaces must be compiled beforehand into platform and language-specific stubs which hide all of the details involved in talking to the server over the network, including marshaling and unmarshaling of arguments and results, and network transport specifics. From the programmer's perspective, the methods of a particular server appear as local methods belonging to that server's object reference, and can be invoked in the same way as any other method in the language. This ease of use is the defining characteristic of the SII.

The Dynamic Invocation Interface (DII) is more difficult to use, but far more general. When using the DII, server methods do not appear conveniently as methods on local objects, but also no pre-compiled stubs are required to invoke operations. Instead, the DII provides a uniform interface for a client to invoke any operation on any object by explicitly specifying the desired operation and the parameters to it. The programmer must also explicitly start the invocation, and possibly manually retrieve any operation results. This process is undeniably tedious, but has the appeal from an ORB implementor's perspective of requiring less development to give CORBA programmers access to all available

objects. Also note that this generality allows an SII to be built with relative ease by writing a stub-generating IDL compiler which utilizes the underlying DII mechanisms.

The Curl ORBlet currently supports only the DII, both because of its generality and because the DII does not require a full CORBA Curl language mapping. It is expected that a stub-generator will be built on top of the DII at some later point, after the language mapping has been finalized. Until then, the DII can provide full, albeit somewhat tedious, access to the distributed object universe.

2.2 Object References

An object reference reliably names a unique CORBA object. Object references can be obtained from a number of sources, but most will come from the naming or trader services. The naming service is just a directory service, supplying object references which match particular names. The trader service is a full-fledged object location service which allows searches for servers to be performed on the basis of keyword arguments. Object references can be used to obtain a variety of information about an object, as well as to invoke methods on the object in the SII. As discussed in the next section, object references are used in the DII to generate dynamic requests to objects. In general, object references are valid only to the ORB which issued them. The Curl ORBlet overcomes this problem by relying on the interoperable object references (IOR) defined as part of the interoperability standard which it uses to communicate with other ORBs. Object references are implemented in Curl by the `Object` class. `Object` currently supports only the `create_request` object reference method.

2.3 Requests

In the DII, requests for services are actually delivered from a client to a server by the `Request` Curl object. A `Request` hides all of the mechanics of the delivery, and

together object references and Requests eliminate any necessity for the user to know about where a server is physically located in either a network or platform sense. The process of invoking a server method is now purely logical.

Requests are created through the create_request method of the server's object reference. The user specifies to create_request the operation to be invoked and the context and arguments for that operation, and gets back a Request object which knows how to invoke the specified operation on that particular server. If no arguments are specified, the programmer must use the Request object's add_arg method to specify the arguments one by one. Whichever method is used to specify the operation's arguments, they must be specified in the same order in which they appear in the operation's Interface Definition Language (IDL) specification. All operation arguments are specified through the use of NamedValues. A NamedValue is a structure which pairs a parameter name with a CORBA any value and a flag denoting the type of the parameter — either in, out, or inout. [1]

All Request methods return an unsigned status code, with zero indicating success and any other value indicating an error. The specific value is meant to serve as a diagnostic, giving more information about the exact nature of the error on a method-specific basis.

2.4 Invoking Requests

After the user has finished building the request, he may invoke it either synchronously or asynchronously. In both types of invocations, the result of the call is left in a NamedValue parameter specified in the original call to create_request.

2.4.1 Synchronous Invocations

Request provides the blocking method invoke to invoke operations synchronously. invoke guarantees that when it returns, the operation has completed success-

1. *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, p. 4-5.

fully, and that the operation result has been left in the `result` parameter specified to `create_request`. As well, all `out` or `inout` parameters in the operation's argument list have been properly updated.

If `invoke` encounters an error somewhere in the invocation process or receives an exception from the server, it will throw an exception identifying the problem. Local exceptions and server-thrown system exceptions are handled correctly, but there is currently a recognized problem with the DII regarding operation-specific exceptions. In the current specification, not enough information is provided to the `Request` object to guarantee that operation-specific exceptions can be unmarshaled without consulting an interface repository, a feature the Curl ORBlet is not guaranteed to have access to. Therefore, if the server throws an operation-specific exception, the system passes up a `USER_EXCEPTION` exception containing the unmarshaled, CDR-encoded exception returned from the operation (see Chapter 3 for a discussion of the Common Data Representation). The user can then use the exception's `TypeCode` to unmarshal the exception.

This solution violates the abstraction barrier between the user's invocation of an operation and the underlying mechanics of how that operation is actually invoked, but it is the simplest solution which insures that no information is lost to the user. There are currently efforts underway to remedy the problems with the DII specification. As soon as these efforts are complete, the abstraction violation should be fixed and all exceptions unmarshaled at or below the `Request` level.

2.4.2 Asynchronous Invocations

The non-blocking `Request` method `send` provides asynchronous invocation abilities. `send` takes a single argument which tells whether or not a response from the operation is expected, and makes no guarantees about the state of the invocation or the state of the passed parameters after it returns. Instead, if the user expects a response, he must

explicitly check on the status of the invocation using the `get_response` method of the `Request` object. In most cases, `get_response` returns a status code indicating whether or not the operation has been completed. If it has completed, then it is guaranteed that the return value of the operation has been left in the `result` parameter passed to `create_request`, and that all `out` and `inout` parameters in the argument list have been updated. If an exception was generated by the operation, then a local exception is generated and thrown by `get_response`. The handling of operation-specific exceptions is the same as in the synchronous case.

Chapter 3

The General Inter-ORB Protocol

3.1 The General Inter-ORB Protocol

Every object in the CORBA universe dwells in some ORB domain, accessible through that ORB to all the other objects in that domain. For most distributed object applications, this arrangement is not a problem. But the fact that the Curl ORBlet could be running on any browser anywhere on the Internet places it essentially outside the domain of any ORB, so it must utilize some other mechanism for communicating with server objects. Fortunately, CORBA defines a protocol which allows objects in one ORB's domain to utilize the services of objects in a different ORB domain. Since the Curl ORBlet essentially provides a mini-domain for its clients, it uses this interoperability protocol to provide those clients with access to objects in the domains of other ORBs.

CORBA defines the General Inter-ORB Protocol (GIOP) as a standard way to allow objects in different ORB domains to communicate across a network using a general connection-oriented transport protocol. To support this communication, GIOP defines a small set of messages which may be exchanged and a common data representation for use in those messages. For actual use, GIOP must be mapped to specific transport protocols. CORBA currently defines only the mapping of GIOP onto TCP/IP, known as the Internet Inter-ORB Protocol (IIOP).

3.2 GIOP Transport Layers

GIOP defines a small set of criteria that must be met by any transport protocol onto which it is to be mapped. Intuitively, the criteria roughly correspond to whether or not the protocol is equivalent in functionality to TCP/IP. First, the protocol must be reliable and connection-oriented. Then the ORB must be able to view the transport layer which

implements the protocol as a virtual byte stream without any restrictions on message size or requirements for fragmentation. Third, there should be some provision for notification of unexpected connection loss. Finally, the manner of initiating connections must be similar to the connection model of TCP/IP. Any transport protocol which provides these features is capable of supporting GIOP.[1]

3.3 Common Data Representation

The unit of communication in GIOP is an octet stream. The CORBA standard defines an octet stream as "an arbitrarily long sequence of eight-bit values (octets) with a well-defined beginning."[2] GIOP defines two types of octet streams — encapsulations and GIOP messages. All data is encoded into octet streams according to the Common Data Representation (CDR) transfer syntax, which defines low-level representations for all of the data types discussed in Chapter 5. Representations in the CDR syntax may be either big or little endian, so that two machines of the same byte-ordering should not need to perform any byte-swapping on the messages they exchange. Furthermore, to allow for efficient handling by architectures which enforce data alignment restrictions, all CDR-encoded data must be aligned on boundaries natural to the type of the data. For example, a CORBA long integer, which takes up four bytes, must be aligned in an octet stream on a word boundary.[3]

In the Curl ORBlet, CDR-encoded octet streams are implemented by the `CDR-buffer` class. `CDR-buffer` provides methods to properly encode and decode all data types, as well as methods to encode or decode values based on a `TypeCode` (see Chapter 5 for details on `TypeCodes`). Alignment restrictions are handled automatically by `CDR-buffer`. To maintain portability, a `CDR-buffer` does not assume any particular byte-

1. *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, p. 12-3.
2. Ibid., p. 12-4.
3. Ibid., p. 12-2.

ordering for its host machine, but relies instead on the class's `endian` attribute to determine in which byte-order it should encode or interpret octet stream data.

While this independence from the host architecture's data representation has advantages for portability, it also has some negative efficiency implications. Instead of being able to copy native representations of data directly into a message and setting the byte-ordering of the message appropriately, the representations must be explicitly computed from the value of the data since the ORBlet cannot depend on a particular underlying representation. A possible future optimization might place the `CDR-buffer` code directly into the kernel so that it could be written to run natively and take advantage of the underlying architecture's representation of the data.

3.3.1 Encapsulations

As mentioned above, GIOP defines two types of octet streams, encapsulations and messages. Encapsulations are octet streams into which data may be marshaled, but which are independent of any particular communication context. GIOP uses encapsulations in several spots where it is desirable that data not be required to be completely unmarshaled before handling. For example, the parameters of a CDR-encoded `TypeCode` are encoded as an encapsulation. Encapsulations are represented as unbounded octet sequences in which the first octet of the stream determines the byte order in which the rest of the stream is encoded. The Curl ORBlet provides the `CDR-encapsulation` class as an implementation of encapsulations.

3.4 GIOP Messages

In GIOP, all communication between ORBs takes place through GIOP messages. There is a small set of messages, seven in total, which among them allow ORBs to cooperate on the full range of standard services. All GIOP messages have a standard header which identifies the protocol (GIOP) and version at use in the message, the byte-ordering of the mes-

sage, the type of the message, and the length in bytes of the message. In addition, particular messages may have particular headers giving more information about the message's contents.[1]

In the Curl ORBlet, the class `GIOP-msg` descends from `CDR-buffer`, and provides methods to extract standard header information. For every supported message type, there is a class which descends from `GIOP-msg` which provides methods for extracting message-type specific data. When a connection is established, a thread is spun off to wait for a response. When one comes, the textual data received from the network port is cast into a message of the appropriate type based on the message-type value in the GIOP header.

3.4.1 Request

Request messages are always sent from client to server and are used by one ORB to request that an operation be invoked on an object in another ORB. The message includes a key identifying the object on which the operation is to be invoked, a string identifying that operation, any parameters to be passed to the operation, and optionally a context for the operation. Request messages are implemented by the `GIOP-request-msg` class in the Curl ORBlet.

3.4.2 Reply

Reply messages are always sent by a server in response to a request message. They contain a service context being passed back to the client, a reply status indicating the outcome of the operation, and a body which contains data appropriate to the outcome. If the operation completed successfully, the body of the message contains the result of the operation followed by all `out` or `inout` parameter values. If an exception occurred, the body of the message contains the exception. The third possibility is that the server sent back a location forward, in which case the body of the message contains an object reference

1. *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, p. 12-15.

pointing to the actual location of the requested server. In the Curl ORBlet, the `GIOP-reply-msg` class implements reply messages.

3.4.3 Others

GIOP defines several other types of messages for various purposes. There is a CloseConnection message which the server uses to indicate that it is shutting down the transport connection to the client. There is also a MessageError message which may be sent by either client or server to indicate that a badly formed message was received. The final two message types are for requesting location information on objects. These last two types are not supported in the Curl ORBlet.

3.5 The Internet Inter-ORB Protocol

Currently TCP/IP is the only transport protocol onto which GIOP has been mapped. The TCP/IP mapping is known as the Internet Inter-ORB Protocol (IIOP). IIOP defines a format for specifying the locations of objects accessible over TCP/IP and some connection usage rules.[1]

1. *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, p. 12-27.

Chapter 4

Request Implementation

4.1 Implementation Overview

The `Request` class is only the tip of the Curl ORBlet iceberg. While it forms the entirety of the user's view of the request invocation process, there is much more going on behind the scenes. Arguments must be marshaled, requests delivered, and results unmarshaled, all in a portable and consistent manner. Furthermore, although the ORBlet currently only knows how to invoke methods on CORBA objects over TCP/IP, it is likely that in the future users will want to invoke requests through other pathways. For example, the CORBA specification includes a protocol for allowing objects in the CORBA universe to interoperate with COM objects. Given the widespread availability of COM objects, the ability to use them from within a Curl client could be very useful. An architecture which allows easy addition of new pathways of invocation such as this one is therefore preferable to one that relies on TCP/IP even at its highest layers.

4.2 Profiles

The Curl ORBlet uses the time-honored approach of layering and the object-oriented principle of polymorphism to build an architecture which supports the easy addition of new invocation pathways. At the top level of this hierarchy is the `Profile` class. A `Profile` encapsulates all knowledge about how to access a particular object through a certain pathway. For objects that can be accessed via multiple pathways, there are multiple `Profiles`. An object reference contains all the `Profiles` which belong to its object, and when a `Request` is created from the object reference it is given a `Profile` which it uses to communicate with the object.

The `Profile` provides a standard interface for the `Request` object to invoke requests on the server. Methods are provided for invoking requests both synchronously and asynchronously. Both methods take as arguments the operation to be invoked, a boolean variable indicating whether or not a response is expected, and a list of parameters, as well as places to leave any result, any exception that might be raised, and a status variable which indicates when the operation is complete and what the outcome was.

Upon return the synchronous method, `InvokeBlockedRequest`, leaves a result status in its `status` parameter. If this status indicates successful completion of the operation, then `InvokeBlockedRequest` guarantees that any return value has been left in `result`, and all `out` and `inout` parameters in the parameter list have been updated. If the status indicates that an exception was thrown, then the exception object has been left in the user-specified location. The same rules apply to the asynchronous method, `InvokeNonBlockedRequest`, except that the status variable will be set to `NOT_DONE` until the operation is completed.

Specific methods of delivery are added to the system by creating classes that inherit from `Profile`, implementing `InvokeBlockedRequest` and `InvokeNonBlockedRequest` as appropriate for the specific delivery method. Currently the only delivery method supported is the General Inter-ORB Protocol.

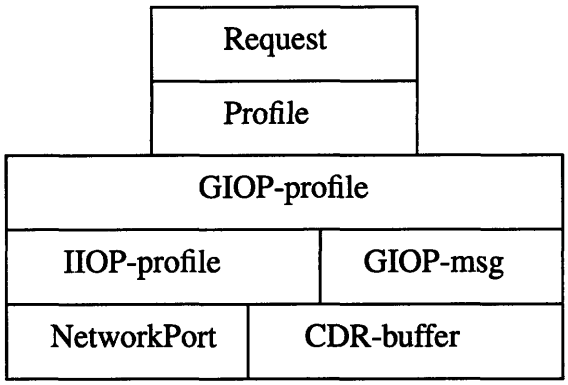

Figure 4.1: The Curl ORBlet request invocation hierarchy

4.3 GIOP Profiles

The GIOP request invocation functionality discussed in Chapter 3 is made available to the request invocation hierarchy through the `GIOP-profile` class. `GIOP-profile` descends from `Profile` and provides incomplete implementations of `InvokeBlockedRequest` and `InvokeNonBlockedRequest` which use GIOP messages to communicate the request to the server. The implementations are incomplete because they rely on unimplemented methods for interfacing with the underlying transport, including methods to open a connection, send a GIOP message, and receive a GIOP message. To complete the implementation, descendants of `GIOP-profile` implement these methods for specific transports.

The availability of an asynchronous invocation mechanism in the DII implies the presence of some sort of threading at lower layers of the invocation hierarchy. `GIOP-profile` is the layer where that threading takes place. Both the synchronous and asynchronous invocation methods spawn listener threads that wait for results to come back and then take actions appropriate to the type of the received message. When this listener thread is finished, it changes the `GIOP-profile`'s `status` attribute to signal the

invoking thread that it is done. The synchronous invocation method waits until the listener thread finishes before returning, while the asynchronous method returns as soon as the thread has been spun off and the request message sent. Note that the current `GIOP-profile` implementation supports at most one outstanding request at any time.

4.4 IIOP Curl Profiles

The implementation of the Curl ORBlet's request invocation mechanism is completed in the `IIOP-profile` class. `IIOP-profile` descends from `GIOP-profile`, and provides TCP/IP-based implementations of the transport methods defined in its parent class using the Curl kernel's `NetworkPort` mechanism.

Chapter 5

Type Mappings

5.1 CORBA types

CORBA is designed to allow diverse objects running on diverse platforms and written in diverse source languages to utilize each other's services. To mask all these differences, all CORBA objects declare their interfaces in a platform-neutral, purely declarative Interface Definition Language (IDL) which is derived from C++. An IDL interface declaration completely specifies the object's interface, but specifies nothing about the object's implementation, including source language or platform. IDL specifies a variety of types that can be used in interface declarations, and in order to maintain consistency with the rest of the universe of CORBA objects, a mapping into Curl for these types must be defined before the request invocation mechanism described in Chapter 4 will function correctly.

The available IDL types fall into four distinct categories of varying complexity. At the simplest level, the usual quanta of data make up the basic IDL types — short, long, signed and unsigned integers, both normal and double precision floating point types, a boolean type, and a character type as well as an 8-bit octet quantity that is guaranteed not to undergo any conversions in transmission. In turn, these basic types form the foundation that the other three type categories build on. The constructed types package data of other types in various ways, and include IDL structures, discriminated unions, and enumerations. Template types allow definition of new types based on a set of template parameters, and include a string type and a dynamic array type. Static arrays are similar to C's arrays.[1]

1. *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, p. 3-19.

IDL type	Curl type
char	charP
octet	octetP
boolean	boolP
long	longP
unsigned long	ulongP
short	shortP
unsigned short	ushortP
float	floatP
double	floatP
struct	class
union	class
enum	ulongP
sequence	Sequence
string	String
array	array
exception	Exception
Object Reference	Object
TypeCode	TypeCodeP
any	CORBA_Any
Principal	Principal

Table 5.1: IDL to Curl type mapping

5.2 Type Mapping Overview

The implementation of the Curl ORBlet places certain functional requirements on the mappings from IDL to Curl. All of the mapped types are required to support dynamic creation of instances using the Curl `new` function with no additional required arguments, although additional optional arguments are fine. This requirement stems from the need to

dynamically generate new instances of types when decoding CDR-encoded sequences. The only exception to this rule are arrays (see section 5.6). Furthermore, to support CDR decoding all types must support call by reference.

5.3 Basic Types

The basic IDL types cannot be mapped directly to the basic Curl types because new instances of basic Curl types cannot be created with the `new` function, nor can basic types be passed by reference. Instead, a layer of indirection must be added to provide the functionality required from the types. To provide this indirection, each basic IDL type maps to a Curl wrapper class with a single member, `val`, which stores the basic type value. For example, the IDL `char` type maps to the following Curl class:

```
{define-class charP {}
  val:char
}
```

All the wrapper classes have names formed by appending "`P`" to the type's IDL name.

5.4 Constructed Types

5.4.1 Structures

Similar to a `struct` in C, an IDL `struct` is a collection of data members which can each be separately accessed. As one would expect, a `struct` maps directly to a Curl class of the same name. For each member of the structure, there is a class member of the same name whose type is determined by the mapping of the structure member's type into Curl. Note that the ordering of the class members does not have to be the same as in the type's IDL definition. An `init` method must be provided to initialize all the members and insure there are no `void` references.

As an example, consider the following IDL definition:

```
struct Foo {
  long fee;
  char initial;
```

```
  boolean paid;
}
```

This IDL definition maps to the following Curl class:

```
{define-class Foo {}
  fee:longP
  initial:charP
  paid:booleanP

  {define {init}
     {set self.fee {new longP}}
     {set self.name {new charP}}
     {set self.paid {new booleanP}}}
}
```

Additional functions may be defined in the class by the user, and the `init` method may take optional arguments.

5.4.2 Unions

An IDL `union` is similar to the `union` type in C, but whereas in C the type currently held in the `union` must be explicitly tracked by the programmer, IDL `unions` are discriminated using a mechanism similar to C's `switch` statement. A discriminator type is specified in the `union` declaration, and each `union` member is assigned a label value of the same type as the discriminator. When the value of the `union`'s discriminator matches the label of a particular member, then that member becomes the current value of the `union`. A `default` label may be specified for at most one member. When the value of the discriminator fails to match any of the other cases, this `default`-labeled value becomes the value of the `union`.

As with `structs`, a `union` maps to a Curl class of the same name. For each member of the `union`, there is a class member of the `union` member's mapped type with the same name. The class must have a member named "_d" of the `union` discriminator's mapped type which serves as the discriminator value. To select which member of the `union` is currently active, the user must directly set the `_d` member. To access the value

of the union, the class must provide a function named "_value" to examine the discriminator and return the appropriate member. In addition, the class must provide a function named "_index" which returns the zero-based index of the currently selected member. This index must reflect the member's location in the IDL union definition, so that it can be used in the TypeCode functions member_type, member_label, and member_name to retrieve TypeCode information about the currently selected member. This functionality is used, for example, in encoding a union into the Common Data Representation. As with structures, an init method must also be provided to initialize all the members so there are no void references in the instance.

As an example, consider the following IDL union definition, where Foo is the structure defined in the struct example above:

```
union Data switch (char) {
  case 'n': long num;
  case 'l': char letter;
  case 'b': boolean pred;
  default: Foo info;
}
```

This IDL definition maps to the following Curl class:

```
{define-class Data {}
  _d:charP
  num:longP
  letter: charP
  pred:boolP
  info:Foo

  {define {init}
     {set self._d {new charP}}
     {set self.num {new longP}}
     {set self.letter {new charP}}
     {set self.pred {new boolP}}
     {set self.info {new Foo}}}

  {define {_value}:any
     {cond {{= self._d.val #\n} {return self.num}}
           {{= self._d.val #\l} {return self.letter}}
           {{= self._d.val #\b} {return self.pred}}
           {else {return self.info}}}}
```

```
    {define {_index}:ulong
       {cond {{= self._d.val #\n} {return 0}}
             {{= self._d.val #\l} {return 1}}
             {{= self._d.val #\b} {return 2}}
             {else {return 3}}}}
}
```

As with `structs`, additional methods may be defined in the class.

5.4.3 Enumerations

IDL enumerations are almost identical to C enumerations. Enumerations define a new data type that can take on one of a set of values specified along with the enumeration declaration. In the Curl CORBA system, `enum` types map to the Curl type `ulongP`, with each value declared in the `enum`'s definition becoming a constant equal to that value's zero-based index in the enumeration sequence. Note that this means that enumeration constants have type `ulong` and cannot be passed directly as enumeration values.

As an example, consider the IDL definition:

```
enum fruit {apple, orange, banana, guava};
```

This IDL definition maps to the following Curl code:

```
{define-constant fruit:type=ulongP}
{define-constant apple:ulong=0}
{define-constant orange:ulong=1}
{define-constant banana:ulong=2}
{define-constant guava:ulong=3}
```

As a consequence of this mapping, there is no checking to insure that `enum` variables are only taking on appropriate values. Such checking is left to the user or the stub compiler when one becomes available.

5.5 Template Types

Curl currently lacks support for template or parameterized types. This part of the type mapping is therefore at best tentative, and is expected to evolve as Curl's support for parameterized types improves.

5.5.1 Sequences

IDL `sequences` are one-dimensional dynamic arrays characterized by two parameters, a content type and a maximum length. The length parameter may be left unspecified, in which case the sequence is unbounded. IDL `sequence` types map to descendants of the `Sequence` Curl class. `Sequence` provides functions to add elements (`put`), reference an element (`aref`), remove an element (`remove`), set an already extant element to a new value (`set`), and find out how many elements are currently in the sequence (`length`). `Sequence` also provides both the `content-type` function to return the Curl type of the data the sequence is intended to hold, and the `bound` function to return the maximum length of the sequence, both of which are useful in decoding CDR-encoded streams. Both `put` and `set` check the type of the passed element against the content type to provide a measure of type safety. `put`, `set`, and `aref` all perform bounds checking as well, with `put` checking against the maximum length of the sequence and `set` and `aref` checking against the current length of the sequence.

A particular CORBA `sequence` type would map to a descendant of the Curl `Sequence` class. The descendant class should provide an `init` constructor to initialize the base `Sequence` class with the proper content type and bound. The name of the descendant class is formed from the IDL declaration by starting with the string "`Sequence_`" and appending the type identifiers spaced by underscores followed by the length, if any, separated with an underscore. For example, the IDL type declaration "`sequence<long, 10>`" maps to the Curl class name "`Sequence_long_10`", and the recursive declaration "`sequence<sequence<char> >`" would map to the class name "`Sequence_Sequence_char`".

As an example of the `sequence` mapping, consider the following IDL declaration:

```
sequence<boolean, 15> predicates;
```

This IDL declaration maps to a Curl variable "predicates" whose type is the class Sequence_boolean_15, defined here:

```
{define-class Sequence_boolean_15 {Sequence}
  {define {init}
     {invoke-method Sequence 'init self theType=boolP
                                          maxlen=15}}
}
```

Although IDL syntactically allows recursive type specifications, sequences are the only place where such specifications are allowed. Since sequences can have a length of zero, they can provide a bottoming out of the recursion that none of the other IDL types can provide

5.5.2 Strings

CORBA strings are closely related to sequences. Like sequences, strings are parameterized, but they only take one parameter, a maximum length. As with sequences, this bound can be left unspecified in which case the string is unbounded. The content type of strings is always char. The CORBA specification states that "[s]trings are singled out as a separate type because many languages have special built-in functions or standard library functions for string manipulation."[1] In Curl, the String class is used to map CORBA strings. String provides some generalized string manipulation functions, as well as bounds checking against a maximum length.

As with sequences, particular bounded strings map to descendants of the String class which provide init methods to properly initialize the bounds checking parameters. Unbounded strings map directly to the String class. The name mappings for strings are also similar to that for sequences. A bounded string name just maps to the class name "String_" followed by the length of the string. For example, the name "string<10>" would map to the Curl class name "String_10".

---

1. *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, p. 3-26.

5.6 Arrays

IDL also defines multidimensional static arrays. Array declarations require explicit bounds on each dimension, and their sizes are fixed at compile time. An IDL array maps directly to a Curl `array` whose content type is the mapped type of the IDL array. Arrays are currently the only exception to the rule about types being required to support a call to the `new` function with no additional arguments. Curl array types are specified by the type they store, and not their length, so the length must be specified as an argument to the initializer. Code that relies on a no-argument constructor being available for an arbitrary type should therefore check for array types and handle them as a special case.

5.7 Exceptions

IDL allows an operation to declare `exceptions` that it might raise under unusual conditions. IDL `exceptions` are similar to `structs` in that other data can be aggregated in them. In the Curl CORBA system, all IDL `exceptions` map to descendants of the `Exception` class. The `exception`'s IDL identifier is available as the `_id` member of the `Exception` it maps to. For specific `exceptions`, each data member maps to a member of the same name in the `Exception`. The type of each `Exception` member is the mapping of the corresponding `exception` member's IDL type into Curl.

CORBA defines a small set of standard exceptions that may be returned from any operation. Each exception represents a broad category of errors and carries with it a minor code to give more detail about the exact nature of the error. These minor codes are currently implementation-dependent, although there are efforts underway now to standardize them for the purposes of ORB interoperability. Each system exception also carries a `completion_status` member which indicates whether the operation was definitely completed before the error occurred, definitely not completed, or maybe completed.

5.8 Pseudo-objects

CORBA defines certain objects which are implemented directly by the ORB. These pseudo-objects look like normal CORBA objects, but invocations of their methods do not follow the normal course of method invocation. Because they are implemented by the ORB, they must be created by the ORB upon explicit request from the user through well-defined functions.

5.8.1 TypeCodes

A CORBA `TypeCode` is meta-data which fully describes an IDL type, in fact a `TypeCode` is equivalent to the IDL definition of the type. A `TypeCode` provides methods for deciding what kind of type it describes as well as for accessing all the data about that type. Each kind of constructed type has specific methods for accessing the data about it; if a method is called on a `TypeCode` describing an incompatible kind, then a `BadKind` exception is thrown. In the Curl ORBlet, `TypeCodes` are implemented by the `TypeCode` class. Note that since `TypeCodes` are immutable, they must be passed in a wrapper class just like the basic types.

CORBA specifies that `TypeCode` constants be provided for each of the basic types as well as for unbounded strings. In the Curl ORBlet, these constants are implemented as functions which return a `TypeCode` object of the proper kind. CORBA also defines functions for constructing `TypeCodes` for constructed and template types which are also provided by the ORBlet.

Given the dynamism of the Curl runtime type environment, a pleasing approach to `TypeCodes` would integrate them with Curl runtime types, allowing the dynamic creation of any IDL type. One possible approach would be to create a `TypeCode` class that descended from the `type` class, and added `TCKind` values and the functional interface specified for `TypeCodes`.

5.8.2 Any

CORBA defines an `any` type which, like Curl's `any` type, packages type information along with a value of that type. In the Curl ORBlet, the IDL `any` type maps to the `CORBA_Any` class, which just packages a `TypeCode` with a Curl `any` value.

5.8.3 Object References

The most important pseudo-object from a user's perspective is the object reference. Embodied by the `Object` class, object references are the user's view of CORBA objects. Information about an object is obtained through its reference, and methods of that object are invoked through its reference. The CORBA standard defines a number of methods for the `Object` class, but in the current implementation of the Curl ORBlet, only the methods directly supporting object method invocation have been implemented.

There is a bootstrap problem in obtaining initial object references. CORBA addresses this problem by defining a pair of functions, `list_initial_services` and `resolve_initial_references`, which are intended to be used in tandem to obtain object references for initial services like the naming service, from which further object references can be obtained. The mapping of these functions into the Curl ORBlet is not yet defined, as there is no standard set of services guaranteed to be available, but any solution will probably involve passing a configuration file along with the ORBlet that somehow encodes a set of services and where to go to find them.

5.8.4 Principal

`Principal` pseudo-objects are used to identify callers to the objects they are calling for security purposes. The security service specification will eventually define a standard representation for `Principal` values, but until then they have no standard format. In the Curl ORBlet, Principals map to an empty class of the same name.

Chapter 6

Future Work

Currently the ORBlet provides only the bare minimum of CORBA functionality, but eventually it should evolve into a full-featured ORB. This evolution provides plenty of opportunities for future work.

6.1 Static Invocation Interface

Availability of an SII would greatly simplify the programming model of the Curl ORBlet. Instead of the tedious Request-building process described in Chapter 2, users would be able to directly invoke an object's methods as if they were local. Creation of an SII would require building a compiler which generated Curl object stubs from IDL declarations of the object's interface. These stubs could rely on the already implemented DII, or for a more efficient approach they could directly handle the sending and receiving of GIOP messages themselves.

6.2 Interface Repository

The interface repository forms an integral part of most ORBs, allowing dynamic investigation of object interfaces, but the current Curl ORBlet is oblivious. A model for how to access a repository needs to be defined, and the IDL types declared for the repository need to be compiled to Curl.

6.3 CORBA Services

CORBA defines a number of useful services to aid in object transactions. The ORBlet should eventually provide full support for these services, as they are where the advantage of a distributed object approach becomes most obvious, but currently the ORBlet knows

nothing about them. Providing support would entail defining a mapping into Curl for the interfaces specified by CORBA and implementing that mapping.

6.4 Servers

The Curl ORBlet does not currently support servers. Even though it is not immediately obvious how server functionality could be useful in a browser-based program, when one starts looking at efforts such as the X consortium's Fresco project, many uses become apparent. In Fresco, graphical interface components are CORBA objects which must be able to receive requests in order to function properly.[1] A curl implementation of this user interface would therefore require server support. To provide server support, the server ORB interfaces as defined by CORBA would have to be implemented.

1. see ftp://sgi.com/graphics/fresco/FAQ for more information on Fresco.

Chapter 7

Conclusion

The world of distributed computing is rapidly shifting away from the traditional client-server model and towards an object model which affords systems as a whole many of the same benefits which object-oriented principles have long brought to programming languages. By encapsulating functionality within objects this model makes it possible to build up applications out of discrete components, possibly from different vendors. No longer do the details of an object's implementation get in the way of its general and widespread use. Now the interface is most important instead, and any client which knows the interface has access to the functionality provided by the object.

At the same time, the web and the sophistication of its content are growing at a lightning pace, and with that growth has come an increased demand for sophisticated data and services as well as a realization that the old web models for dynamic content are no longer sufficient. The Common Gateway Interface was never more than an ad hoc add-on to simulate dynamic content, and under the intense usage of today CGI has started to show cracks. A new method of providing services via the web is much needed.

It is not hard to see these two trends converging to create a new network landscape where portable browser-based clients utilize distributed server objects to provide dynamic and sophisticated data and services to users. In this new web, Curl's support for the distributed object system CORBA in the form of the Curl ORBlet greatly enhances its potential as a platform for building distributed applications.

Appendix A

Curl ORBlet Source Code

A.1 ORB/request.curl

```
|| File: ORB/request.curl
|| Defines: The Request and Profile classes

{include "types.curl"}
{include "exceptions.curl"}

|| the Status type is defined for returning result status codes.
|| a Status value of 0 generally indicates success, while anything
|| else generally indicates a failure of some kind with the specific
|| value giving more detail about the exact failure. these failure
|| values are not standard from one function to another, but should
|| be documented with each function.
{define-constant Status:type=ulong}
{define-constant StatusP:type=ulongP}

|| status codes
{define-constant STATUS_OK:Status=0}
{define-constant NOT_DONE:Status=1}
{define-constant UNKNOWN_EXC:Status=1}
{define-constant SYS_EXC_STATUS:Status=2}
{define-constant USER_EXC_STATUS:Status=3}

|| bit-masks for flags
{define-constant INV_NO_RESPONSE:long=#x00000001}

{define Request {}
  target:Profile
  args: NVList
  op:Identifier
  ctx:Context
  result:NamedValue
  status:StatusP
  exc:exceptionP

  {define {init target: CORBA-Profile ctx:Context op:Identifier
args:NVList result:NamedValue}
    {set self.ctx ctx}
    {set self.args args}
    {set self.op op}
    {set self.result result}
    {set self.target target}
    {set self.status {new StatusP val=STATUS_OK}}}

  {define {add_arg name:Identifier argtype:TypeCode
   argval:any argflags:Flags}:Status
```

```
   {let val:CORBA_Any={new CORBA_Any argval argtype}
     {self.args.add-value name val argflags}}}

  {define {invoke invoke_flags:Flags}:Status
    {let repl-expected:bool={bit-and invoke_flags INV_NO_RESPONSE}
      || initialize
      {set self.status {new StatusP val=NOT_DONE}}
      {set self.exc {new exceptionP}}
      || invoke request
      {self.target.InvokeBlockedRequest self.op repl-expected
self.args self.ctx
self.status self.exc}
      || check result status
      || if we got an exception then throw it, it should have
      || been created for us by the profile
      {if {or {= self.status.val SYSTEM_EXCEPTION}
      {= self.status.val USER_EXCEPTION}}
  {throw exc.val}}
      || otherwise we're OK
      {return STATUS_OK}}}

  {define {send invoke_flags:Flags}:Status
    {let repl-expected:bool={bit-and invoke_flags INV_NO_RESPONSE}
      || initialize
      {set self.status {new StatusP val=NOT_DONE}}
      {set self.exc {new exceptionP}}
      || invoke request
      {self.target.InvokeNonBlockedRequest self.op repl-expected
   self.args self.ctx
   self.status self.exc}
      || if we made it to this point then we're OK, at least for now
      {return STATUS_OK}}}

  {define {get_response response_flags:Flags}:Status
    || check to see if we've gotten a reply
    {if {= self.status.val NOT_DONE}
{return NO_REPLY}}
    || check for exceptions, if we got one throw it up, it should have
    || been created for us by the profile
    {if {or {= self.status.val SYSTEM_EXCEPTION}
    {= self.status.val USER_EXCEPTION}}
  {throw exc.val}}
    || otherwise we've returned and everything is OK
    {return STATUS_OK}}

}

{define-class Profile {}
  {define {InvokeBlockedRequest op:text
repl-expected:bool
params:NVList
result:NamedValue
context:Context
status:StatusP
```

```
exc:exceptionP}:Status
    }

  {define {InvokeNonBlockedRequest op:text
   repl-expected:bool
   params:NVList
   result:NamedValue
   context:Context
   status:StatusP
   exc:exceptionP}:Status
    }
}
```

A.2 ORB/nvlist.curl

```
|| File: ORB/nvlist.curl
|| Defines: NamedValue and NVList, both of which are data structures
||          used to specify arguments to the DII

{include "typecode.curl"}
{include "debug.curl"}

{define-constant Status:type=int}

{define-class NamedValue {}
  name:Identifier
  argument:CORBA_Any
  arg_modes:Flags

  {define {init name:Identifier argument:CORBA_Any item_flags:Flags}
    {set self.name name}
    {set self.argument argument}
    {set self.arg_modes item_flags}}


  {define {print}
    {if {not {void? self.name}}
{begin
  {printout "name: "}
  {println {self.name.toText}}}}
    {if {not {void? self.argument}}
{begin
  {printout "arg type: "}
  {println self.argument.item_type}
  {printout "arg val: "}
  {println self.argument.item}}}
    {printout "arg flags: "}
    {println self.arg_modes}}

}

{define-class NVListElement {}
  value:NamedValue
```

```
  next:NVListElement
  {define {init name:Identifier argument:CORBA_Any item_flags:Flags}
    {set self.value {new NamedValue name argument item_flags}}
    {set self.next void}}
}

{define-class NVList {}
  the_list:NVListElement
  count:int

  {define {init}
    {set self.the_list void}
    {set self.count 0}}

  {define {print}
    {printout "length: "}
    {println self.count}
    {let curr:NVListElement=self.the_list
      {while {not {void? curr}}
     {curr.value.print}
     {set curr curr.next}}}}

  {define {get_count}:int
    {return self.count}}

  {define {prepend-named-value val:NamedValue}
    {let valElem:NVListElement={new NVListElement void void 0}
      {set valElem.value val}
      {set valElem.next self.the_list}
      {set self.the_list valElem}}}

  {define {add-item name:Identifier item_flags:Flags}
    {self.add-value name void item_flags}}

  {define {add-value name:Identifier argument:CORBA_Any
item_flags:Flags}
    {let member:NVListElement={new NVListElement name argument
item_flags}
 curr:NVListElement=self.the_list
      {if {void? self.the_list}
  {set self.the_list member}
  {begin
    {while {not {void? curr.next}}
   {set curr curr.next}}
    {set member.next void}
    {set curr.next member}}}
      {set self.count {+ 1 self.count}}}}

  {define {item index:int}:NamedValue
    {if {> index self.count}
{throw {new BoundsEx}}}
    {let i:int = 0
 curr:NVListElement = self.the_list
 {while {and {!= i index} {not {void? curr}}}
```

```
{set curr curr.next}
{set i {+ i 1}}}
 {if {void? curr}
     {throw {new BoundsEx}}
     {return curr.value}}}}

}
```

A.3 ORB/ior.curl

```
|| File: ORB/ior.curl
|| Defines: the Object class which is the Curl ORBlet's implementation
||          of interoperable object references

{include "types.curl"}
{include "request.curl"}

{define-constant ProfileId:type=ulong}
{define-constant TAG_INTERNET_IOP:ProfileId=0}
{define-constant TAG_MULTIPLE_COMPONENTS:ProfileId=1}

{define RequestP {}
  val:Request
}

{define-class TaggedProfile {}
  tag:ProfileId
  profile:OctetSeq

  {define {init tag:ProfileId=TAG_INTERNET_IOP profile:Profile=void}
    {set self.tag tag}
    {set self.profile profile}}
}

{define {TC_TaggedProfile}:TypeCode
  {let mems:StructMemberSeq={new StructMemberSeq}
    {mems.put {new StructMember "tag" {TC_ulong}}}
    {mems.put {new StructMember "profile" {TC_sequence}}}
    {return {create_struct_tc {text->String ""} {text->String "Tagged-
Profile"}
      mems}}}}

{define-class Sequence_TaggedProfile {Sequence}
  {define {init}
    {invoke-method Sequence 'init self theType=TaggedProfile max-
len=0}}
}

{define {TC_sequence_TaggedProfile}
  {return {create_sequence_tc 0 {TC_TaggedProfile}}}}

{define-constant TPSeq:type=Sequence_TaggedProfile}
```

```
{define-class Object {}
  type_id:String
  profiles:TPSeq

  {define {init id:String={new String}}
    {set type_id id}
    {set profiles {new TPSeq}}}

  {define {add-profile tp:TaggedProfile}
    {self.profiles.put tp}}

  {define {create_request ctx:Context operation:Identifier
arg_list:NVList
  result:NamedValue request:RequestP
  Flags:req_flags}:Status
    || find a profile that will work
    {let count:ulong={self.profiles.count}
         i:ulong=0
 profile:Profile=void
      {while {and {< i count} {void? profile}}
|| IIOP is the only protocol we support right now
{if {= {self.profiles.aref i}.tag TAG_INTERNET_IOP}
    {set profile {make-iiop-profile {OctetSeq->String
     {self.profiles.aref i}.profile}}}}
{set i {+ i 1}}}
      || now create request object
      {set request.val {new Request profile ctx operation args result}}
      {return STATUS_OK}}}
}
```

A.4 ORB/types.curl

```
|| File: ORB/ior.curl
|| Defines: the Object class which is the Curl ORBlet's implementation
||          of interoperable object references

{include "types.curl"}
{include "request.curl"}

{define-constant ProfileId:type=ulong}
{define-constant TAG_INTERNET_IOP:ProfileId=0}
{define-constant TAG_MULTIPLE_COMPONENTS:ProfileId=1}

{define RequestP {}
  val:Request
}

{define-class TaggedProfile {}
  tag:ProfileId
  profile:OctetSeq
```

```
  {define {init tag:ProfileId=TAG_INTERNET_IOP profile:Profile=void}
    {set self.tag tag}
    {set self.profile profile}}
}

{define {TC_TaggedProfile}:TypeCode
  {let mems:StructMemberSeq={new StructMemberSeq}
    {mems.put {new StructMember "tag" {TC_ulong}}}
    {mems.put {new StructMember "profile" {TC_sequence}}}
    {return {create_struct_tc {text->String ""} {text->String "Tagged-
Profile"}
      mems}}}}

{define-class Sequence_TaggedProfile {Sequence}
  {define {init}
    {invoke-method Sequence 'init self theType=TaggedProfile max-
len=0}}
}

{define {TC_sequence_TaggedProfile}
  {return {create_sequence_tc 0 {TC_TaggedProfile}}}}

{define-constant TPSeq:type=Sequence_TaggedProfile}

{define-class Object {}
  type_id:String
  profiles:TPSeq

  {define {init id:String={new String}}
    {set type_id id}
    {set profiles {new TPSeq}}}

  {define {add-profile tp:TaggedProfile}
    {self.profiles.put tp}}

  {define {create_request ctx:Context operation:Identifier
arg_list:NVList
  result:NamedValue request:RequestP
  Flags:req_flags}:Status
    || find a profile that will work
    {let count:ulong={self.profiles.count}
         i:ulong=0
 profile:Profile=void
      {while {and {< i count} {void? profile}}
|| IIOP is the only protocol we support right now
{if {= {self.profiles.aref i}.tag TAG_INTERNET_IOP}
    {set profile {make-iiop-profile {OctetSeq->String
     {self.profiles.aref i}.profile}}}}
{set i {+ i 1}}}
      || now create request object
     {set request.val {new Request profile ctx operation args result}}
      {return STATUS_OK}}}
}
```

A.5 ORB/exceptions.curl

```
|| File: ORB/exceptions.curl
|| Defines: the Exception class as well as all the system exceptions

{include "types.curl"}

{define-class ExceptionP {}
  val:Exception

  {define {init val:Exception={new Exception}}
    {set self.val val}}
}

{define-class Exception {}
  _id:String

  {define {init id:String={new String}}
    {set self._id {new String}}
    {self._id.concatenate id}}
}

|| standard exceptions
|| these are all defined in section 3.15.1 of
|| the IDL chapter of the  CORBA 2.0 spec

{define-constant completion_status:type=ulongP}
{define-constant COMPLETED_YES:ulong=0}
{define-constant COMPLETED_NO:ulong=1}
{define-constant COMPLETED_MAYBE:ulong=2}

{define-class SystemException {Exception}
  minor:ulong
  completed:completion_status

  {define {init id:String={new String}
status:completion_status=COMPLETED_MAYBE
minor:ulong=0}
    {invoke-method Exception 'init self id}
    {set self.completed status}
    {set self.minor minor}}
}

|| an unrecognized user exception was thrown
{define-class USER_EXCEPTION {Exception}
  buff:CDR-buffer || the CDR-encoded user exception

  {define {init buff:CDR-buffer={new CDR-buffer}}
    {invoke-method Exception 'init self id={new String
txt="USER_EXCEPTION"}}
    {set self.buff buff}}
}
```

```
|| the unknown SystemException
{define-class UNKNOWN {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self id={new String
txt="UNKNOWN"}
   minor=minor}}
}

|| an invalid parameter was passed
{define-class BAD_PARAM {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="BAD_PARAM"} minor=minor}}
}

|| dynamic memory allocation failure
{define-class NO_MEMORY {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="NO_MEMORY"} minor=minor}}
}

|| violated implementation limit
{define-class IMP_LIMIT {SystemException}
  {define {init minor:ulong}
    {invoke-method SystemException 'init self
   id={new String txt="IMP_LIMIT"} minor=minor}}
}

|| communication failure
|| minor codes:
{define-constant BAD_REPLY:ulong=0}
{define-constant CONNECTION_CLOSED:ulong=1}
{define-constant ERROR_RECEIVED:ulong=2}

{define-class COMM_FAILURE {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="COMM_FAILURE"} minor=minor}}
}

|| invalid object reference
{define-class INV_OBJREF {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="INV_OBJREF"} minor=minor}}
}

|| no permission for attempted operation
{define-class NO_PERMISSION {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="NO_PERMISSION"} minor=minor}}
```

```
}

|| ORB internal error
{define-class INTERNAL {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="INTERNAL"} minor=minor}}
}

|| error marshalling param/result
{define-class MARSHAL {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="MARSHAL"} minor=minor}}
}

|| ORB initialization failure
{define-class INITIALIZE {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="INITIALIZE"} minor=minor}}
}

|| operation implementation unavailable
{define-class NO_IMPLEMENT {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="NO_IMPLEMENT"} minor=minor}}
}

|| bad typecode
{define-class BAD_TYPECODE {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="BAD_TYPECODE"} minor=minor}}
}

|| invalid operation
{define-class BAD_OPERATION {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="BAD_OPERATION"} minor=minor}}
}

|| insufficient resources for request
{define-class NO_RESOURCES {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="NO_RESOURCES"} minor=minor}}
}

|| response to request not yet available
{define-class NO_RESPONSE {SystemException}
  {define {init minor:ulong=0}
```

```
    {invoke-method SystemException 'init self
   id={new String txt="NO_RESPONSE"} minor=minor}}
}

|| persistent storage failure
{define-class PERSIST_STORE {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="PERSIST_STORE"} minor=minor}}
}

|| routine invocations out of order
{define-class BAD_INV_ORDER {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="BAD_INV_ORDER"} minor=minor}}
}

|| transient failure -- reissue request
{define-class TRANSIENT {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="TRANSIENT"} minor=minor}}
}

|| cannot free memory -- this shouldn't be a problem in Curl
{define-class FREE_MEM {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="FREE_MEM"} minor=minor}}
}

|| invalid identifier syntax
{define-class INV_IDENT {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="INV_IDENT"} minor=minor}}
}

|| invalid flag was specified
{define-class INV_FLAG {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="INV_FLAG"} minor=minor}}
}

|| error accessing interface repository
{define-class INTF_REPOS {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="INTF_REPOS"} minor=minor}}
}

|| error processing context object
```

```
{define-class BAD_CONTEXT {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="BAD_CONTEXT"} minor=minor}}
}

|| failure detected by object adapter
{define-class OBJ_ADAPTER {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="OBJ_ADAPTER"} minor=minor}}
}

|| data conversion error
{define-class DATA_CONVERSION {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="DATA_CONVERSION"} minor=minor}}
}

|| non-existent object, delete reference
{define-class OBJECT_NOT_EXIST {SystemException}
  {define {init minor:ulong=0}
    {invoke-method SystemException 'init self
   id={new String txt="OBJECT_NOT_EXIST"} minor=minor}}
}

|| check if id is the name of a system exception
{define {system-exception? id:String}
  {if {or {id.text-eq? "UNKNOWN"}{id.text-eq? "BAD_PARAM"}
  {id.text-eq? "NO_MEMORY"}{id.text-eq? "IMP_LIMIT"}
  {id.text-eq? "COMM_FAILURE"}{id.text-eq? "INV_OBJREF"}
  {id.text-eq? "NO_PERMISSION"}{id.text-eq? "INTERNAL"}
  {id.text-eq? "MARSHAL"}{id.text-eq? "INITIALIZE"}
  {id.text-eq? "NO_IMPLEMENT"}{id.text-eq? "BAD_TYPECODE"}
  {id.text-eq? "BAD_OPERATION"}{id.text-eq? "NO_RESOURCES"}
  {id.text-eq? "NO_RESPONSE"}{id.text-eq? "PERSIST_STORE"}
  {id.text-eq? "BAD_INV_ORDER"}{id.text-eq? "TRANSIENT"}
  {id.text-eq? "FREE_MEM"}{id.text-eq? "INV_IDENT"}
  {id.text-eq? "INV_FLAG"}{id.text-eq? "INTF_REPOS"}
  {id.text-eq? "BAD_CONTEXT"}{id.text-eq? "OBJ_ADAPTER"}
  {id.text-eq? "DATA_CONVERSION"}{id.text-eq? "OBJECT_NOT_EXIST"}}
      {return true}
      {return false}}}
```

A.6 ORB/ir.curl

```
|| File: ir.curl
|| Defines: A few interface repository classes which are useful in
||          building typecodes.
```

```
{include "typecode.curl"}
{include "types.curl"}

{define-class StructMember {}
  name:Identifier
  type:TypeCode

  {define {init name:Identifier tc:TypeCode}
    {set self.name {name.duplicate}}
    {set self.type {tc.copy}}}

  {define {equal? other:StructMember}:bool
    {return {self.type.equal other.type}}}

  {define {print}
    {printout "name: "}
    {self.name.println}
    {printout "type: "}
    {self.type.print}}

  {define {copy}:StructMember
    {return {new StructMember self.name self.type}}}
}

{define-class UnionMember {}
  name:Identifier
  label:CORBA_Any
  type:TypeCode

  {define {init name:Identifier tc:TypeCode label:CORBA_Any}
    {set self.name {name.duplicate}}
    {set self.type {tc.copy}}
    {printout "UnionMember init: name: "}
    {name.println}
    {set self.label {new CORBA_Any val=label.item
 theType={label.item_type.val.copy}}}
    {self.label.item_type.val.print}}

  {define {equal? other:UnionMember}:bool
    {let val1:word=self.label.item
 val2:word=other.label.item
      {if {not {= val1 val2}}
  {return false}}}
    {return {self.type.equal other.type}}}

  {define {print}
    {printout "name: "}
    {self.name.println}
    {printout "type: "}
    {self.type.print}
    {let val:word=self.label.item
      {printout "label: "}
      {println val}}}
```

```
  {define {copy}:UnionMember
    {return {new UnionMember self.name self.type self.label}}}

}

{define-class StructMemberSeq {Sequence}
  {define {init}
    {invoke-method Sequence 'init self theType=StructMember}}

  {define {copy}:StructMemberSeq
    {let length:ulong={self.length}
      i:ulong=0
      newseq:StructMemberSeq={new StructMemberSeq}
      {while {< i length}
     {newseq.put {{self.aref i}.copy}}
     {set i {+ i 1}}}
      {return newseq}}}

}

{define-class UnionMemberSeq {Sequence}
  {define {init}
    {invoke-method Sequence 'init self theType=UnionMember}}

  {define {copy}:UnionMemberSeq
    {let length:ulong={self.length}
      i:ulong=0
      newseq:UnionMemberSeq={new UnionMemberSeq}
      {while {< i length}
     {newseq.put {{self.aref i}.copy}}
     {set i {+ i 1}}}
      {return newseq}}}

}

{define-class EnumMemberSeq {Sequence}
  {define {init}
    {invoke-method Sequence 'init self theType=Identifier}}

  {define {copy}:EnumMemberSeq
    {let length:ulong={self.length}
      i:ulong=0
      newseq:EnumMemberSeq={new EnumMemberSeq}
      {while {< i length}
     {newseq.put {{self.aref i}.copy}}
     {set i {+ i 1}}}
      {return newseq}}}

}
```

A.7 ORB/typecode.curl

```
|| File: ORB/typecode.curl
|| Defines: TypeCode
||          TypeCode is one of the fundamental building blocks of the
||          dynamic interfaces, as it allows for on-the-fly type
||          specification (TypeCode is meta-data).

{include "types.curl"}
{include "debug.curl"}
{include "ir.curl"}



{define {create_struct_tc id:RepositoryId name:Identifier
  members:StructMemberSeq}:TypeCode
  {let params:TkStructParams={new TkStructParams name id members}
       tc:TypeCode={new TypeCode kind=TK_struct params=params}
    {let count:ulong={members.length}
         i:ulong=0
      {while {< i count}
{let mem:StructMember={params.members.aref i}
  {set mem.type.parent tc}}
{set i {+ i 1}}}}
    {return tc}}}

{define {create_union_tc id:RepositoryId name:Identifier
 discriminator_type:TypeCode
 members:UnionMemberSeq}:TypeCode
  || check disctype to make sure it is a valid label type
  {let disctype:TypeCode=discriminator_type
      {if {not {or {= {disctype.kind}.val TK_long}
   {= {disctype.kind}.val TK_ulong}
   {= {disctype.kind}.val TK_short}
   {= {disctype.kind}.val TK_ushort}
   {= {disctype.kind}.val TK_char}
   {= {disctype.kind}.val TK_boolean}
   {= {disctype.kind}.val TK_enum}}}
  {throw {new BadKindEx}}}}
  {let params:TkUnionParams={new TkUnionParams name id
 members discriminator_type}
       tc:TypeCode={new TypeCode kind=TK_union params=params}
    {let count:ulong={members.length}
         i:ulong=0
      {while {< i count}
{let mem:UnionMember={params.members.aref i}
  {set mem.type.parent tc}}
{set i {+ i 1}}}}
    {return tc}}}

{define {create_enum_tc id:RepositoryId name:Identifier
  members:EnumMemberSeq}:TypeCode
  {let params:TkEnumParams={new TkEnumParams name id members}
    {return {new TypeCode kind=TK_enum params=params}}}}
```

```
{define {create_alias_tc id:RepositoryId name:Identifier
 original_type:TypeCode}:TypeCode
  {let params:TkAliasParams={new TkAliasParams name id original_type}
    {return {new TypeCode kind=TK_alias params=params}}}}

{define {create_exception_tc id:RepositoryId name:Identifier
     members:StructMemberSeq}:TypeCode
  {let tc:TypeCode={create_struct_tc id name members}
    {set tc._kind TK_except}
    {return tc}}}

{define {create_interface_tc id:RepositoryId name:Identifier}:Type-
Code
  {let namemem:StructMember={new StructMember
 {text->String "name"}
 {TC_string}}
    idmem:StructMember={new StructMember
    {text->String "id"}
    {TC_string}}
    defmem:StructMember={new StructMember
    {text->String "defined_in"}
    {TC_string}}
    vermem:StructMember={new StructMember
     {text->String "version"}
     {TC_string}}
    intmem:StructMember={new StructMember
     {text->String "base_interfaces"}
     {create_sequence_tc 0 {TC_string}}}
    members:StructMemberSeq={new StructMemberSeq}
    {members.put namemem}
    {members.put idmem}
    {members.put defmem}
    {members.put vermem}
    {members.put intmem}
    {return {create_struct_tc id name members}}}}

{define {create_string_tc bound:ulong}:TypeCode
    {return {new TypeCode kind=TK_string params={new TkStringParams
bound}}}}

{define {create_sequence_tc bound:ulong element_type:TypeCode}:Type-
Code
  {let params:TkSequenceParams={new TkSequenceParams element_type
bound}
      tc:TypeCode={new TypeCode
kind=TK_sequence
params=params}
    {set params.content_type.parent tc}
    {return tc}}}

{define {create_recursive_sequence_tc bound:ulong offset:ulong}:Type-
Code
  {return {create_sequence_tc bound {new TypeCode
```

```
 kind=TK_indirect
 params={new TkIndirectParams
     offset}}}}}

{define {create_array_tc length:ulong element_type:TypeCode}:TypeCode
  {let params:TkArrayParams={new TkArrayParams element_type length}
      tc:TypeCode={new TypeCode
kind=TK_array
params=params}
    {set params.content_type.parent tc}
    {return tc}}}

{define {create_objref_tc id:RepositoryId name:Identifier}:TypeCode
  {return {new TypeCode kind=TK_objref params={new TkObjRefParams id
name}}}}


{define-constant TCKind:type=ulongP}
{define-constant TK_null:ulong=0}
{define-constant TK_void:ulong=1}
{define-constant TK_short:ulong=2}
{define-constant TK_long:ulong=3}
{define-constant TK_ushort:ulong=4}
{define-constant TK_ulong:ulong=5}
{define-constant TK_float:ulong=6}
{define-constant TK_double:ulong=7}
{define-constant TK_boolean:ulong=8}
{define-constant TK_char:ulong=9}
{define-constant TK_octet:ulong=10}
{define-constant TK_any:ulong=11}
{define-constant TK_TypeCode:ulong=12}
{define-constant TK_Principal:ulong=13}
{define-constant TK_objref:ulong=14}
{define-constant TK_struct:ulong=15}
{define-constant TK_union:ulong=16}
{define-constant TK_enum:ulong=17}
{define-constant TK_string:ulong=18}
{define-constant TK_sequence:ulong=19}
{define-constant TK_array:ulong=20}
{define-constant TK_alias:ulong=21}
{define-constant TK_except:ulong=22}
{define-constant TK_indirect:ulong=#xffffffff}


|| indirects are semi-visible. for example, CDR has to know about them
|| in order to encode everything correctly. but most user code
shouldn't
|| have to deal with them
{define-class TypeCode {}
  _kind:TCKind
  params:any
  parent:TypeCode

  {define {print}:void
```

```
    || handle the indirect case specially
    {if {= self._kind.val TK_indirect}
{begin
  {printout "kind: "}
  {println self._kind.val}
  {printout "indirect: "}
  {println self.params.offset}
  {return}}}
    {let kind:ulong={self.kind}.val
      {printout "kind: "}
      {println kind}
      || print out id and name for applicable types
      {if {or {= kind TK_struct}{= kind TK_union}{= kind TK_alias}
      {= kind TK_except}{= kind TK_objref}{= kind TK_enum}}
       {let id:RepositoryId={self.id}
 name:Identifier={self.name}
 {printout "id: "}
 {id.println}
 {printout "name: "}
 {name.println}}}

      || print out member count for applicable types
      {if {or {= kind TK_struct}{= kind TK_except}{= kind TK_union}
      {= kind TK_enum}}
        {let
    count:ulong={self.member_count}
    {printout "member count: "}
    {println count}}}

      || print out length for sequences, strings, and arrays
      {if {or {= kind TK_sequence}{= kind TK_array}{= kind TK_string}}
  {begin
    {printout "length: "}
    {println {self.length}}}}

      || print out content type for sequences, aliases, and arrays
      {if {or {= kind TK_sequence}{= kind TK_array}{= kind TK_alias}}
  {let tc:TypeCode={self.content_type}
    {println "------content type------"}
    {tc.print}
    {println "------------------------"}}}

      || print out members for enum
      {if {= kind TK_enum}
  {let count:ulong={self.member_count}
       i:ulong=0
    {while {< i count}
      {let name:Identifier={self.member_name i}
{println "================="}
{printout "member #"}
{println i}
{printout "name: "}
{name.println}
      {set i {+ i 1}}}}}}}
```

```
      || print out members for struct and exception
      {if {or {= kind TK_struct}{= kind TK_except}}
  {let count:ulong={self.member_count}
       i:ulong=0
    {while {< i count}
      {let name:Identifier={self.member_name i}
   tc:TypeCode={self.member_type i}
{println "================="}
{printout "member #"}
{println i}
{printout "name: "}
{name.println}
{println "type: "}
{println "------------"}
{tc.print}
{println "------------"}
      {set i {+ i 1}}}}}}

      || print out info for union
      {if {= kind TK_union}
  {let count:ulong={self.member_count}
       i:ulong=0
       default_index:long={self.default_index}
       disctype:TypeCode={self.discriminator_type}
    {printout "default index: "}
    {println default_index}
    {println "discriminator type: "}
    {disctype.print}
    {while {< i count}
      {let name:Identifier={self.member_name i}
   tc:TypeCode={self.member_type i}
   label:CORBA_Any={self.member_label i}
   val:word=label.item.val
{println "================="}
{printout "member #"}
{println i}
{printout "label: "}
{println val}
{printout "name: "}
{name.println}
{println "type: "}
{println "------------"}
{tc.print}
{println "------------"}
      {set i {+ i 1}}}}}}
      }}


  {define {copy}:TypeCode
    {let tc:TypeCode={new TypeCode}
      {set tc._kind {new TCKind val=self._kind.val}}
      {set tc.parent self.parent}
      {if {not {void? self.params}}
```

```
  {set tc.params {self.params.copy tc}}}
      {return tc}}}

  || initialization
  {define {init kind:ulong=TK_null params:any=void parent:Type-
Code=void}
    {set self._kind {new TCKind val=kind}}
    {set self.parent parent}
    {set self.params params}}

  {define {equal tc:TypeCode}:bool
    {printout "tc:equal: top..."}
    {if {!= self._kind.val tc._kind.val}
{return false}}
    {printout "tc:equal: checking params..."}
    {if {not {void? self.params}}
{return {self.params tc.params}}}
    {return true}}

  {define {kind}:TCKind
    || if indirect, locate ancestor and invoke method on it
    {if {= self._kind.val TK_indirect}
{return {{self.dereference}.kind}}}
    || otherwise just return _kind
    {return {new TCKind val=self._kind.val}}}

  || for tk_objref, tk_struct, tk_union, tk_enum, tk_alias, and
tk_except
  {define {id}:RepositoryId
    || if indirect, locate ancestor and invoke method on it
    {if {= self._kind.val TK_indirect}
{return {{self.dereference}.id}}}
    {if {not {or {= {self.kind}.val TK_objref}
 {= {self.kind}.val TK_struct}
 {= {self.kind}.val TK_union}
 {= {self.kind}.val TK_enum}
 {= {self.kind}.val TK_alias}
 {= {self.kind}.val TK_except}}}
{throw {new BadKindEx}}}
    {if {void? self.params}
{throw {new InitializationEx}}}
    {return self.params.id}}

  || for tk_objref, tk_struct, tk_union, tk_enum, tk_alias, and
tk_except
  {define {name}:Identifier
    || if indirect, locate ancestor and invoke method on it
    {if {= self._kind.val TK_indirect}
{return {{self.dereference}.name}}}
    {if {not {or {= {self.kind}.val TK_objref}
 {= {self.kind}.val TK_struct}
 {= {self.kind}.val TK_union}
 {= {self.kind}.val TK_enum}
 {= {self.kind}.val TK_alias}
```

```
 {= {self.kind}.val TK_except}}}
{throw {new BadKindEx}}}
    {if {void? self.params}
{throw {new InitializationEx}}}
    {return self.params.name}}


  || for tk_objref, tk_struct, tk_union, tk_enum, and tk_except
  {define {member_count}:ulong
    || if indirect, locate ancestor and invoke method on it
    {if {= self._kind.val TK_indirect}
{return {{self.dereference}.member_count}}}
    {if {not {or {= {self.kind}.val TK_struct}
 {= {self.kind}.val TK_union}
 {= {self.kind}.val TK_enum}
 {= {self.kind}.val TK_except}}}
{throw {new BadKindEx}}}
    {if {void? self.params}
{throw {new InitializationEx}}}
    {return {self.params.members.length}}}


  || for tk_struct, tk_union, tk_enum, and tk_except
  {define {member_name index:ulong}:Identifier
    || if indirect, locate ancestor and invoke method on it
    {if {= self._kind.val TK_indirect}
{return {{self.dereference}.member_name index}}}
    {if {not {or {= {self.kind}.val TK_struct}
 {= {self.kind}.val TK_union}
 {= {self.kind}.val TK_enum}
 {= {self.kind}.val TK_except}}}
{throw {new BadKindEx}}}
    {if {void? self.params}
{throw {new InitializationEx}}}
    {if {or {< index 0} {>= index {self.member_count}}}
{throw {new BoundsEx}}}
    || return name of index'th member...
    {if {or {= {self.kind}.val TK_struct}
    {= {self.kind}.val TK_union}
    {= {self.kind}.val TK_except}}
{return {self.params.members.aref index}.name}}
    {return {self.params.members.aref index}}}



  || for tk_struct, tk_union, and tk_except
  {define {member_type index:ulong}:TypeCode
    || if indirect, locate ancestor and invoke method on it
    {if {= self._kind.val TK_indirect}
{return {{self.dereference}.member_type index}}}
    {if {not {or {= {self.kind}.val TK_struct}
 {= {self.kind}.val TK_union}
 {= {self.kind}.val TK_except}}}
{throw {new BadKindEx}}}
```

```
    {if {void? self.params}
{throw {new InitializationEx}}}
    {if {or {< index 0} {>= index {self.member_count}}}
{throw {new BoundsEx}}}
    || return type of index'th member...
    {return {self.params.members.aref index}.type}}

  || for tk_union
  {define {member_label index:ulong}:CORBA_Any
    || if indirect, locate ancestor and invoke method on it
    {if {= self._kind.val TK_indirect}
{return {{self.dereference}.member_label index}}}
    {if {not {= {self.kind}.val TK_union}}
{throw {new BadKindEx}}}
    {if {void? self.params}
{throw {new InitializationEx}}}
    {if {or {< index 0} {>= index {self.member_count}}}
{throw {new BoundsEx}}}
    || return type of index'th member...
    {return {self.params.members.aref index}.label}}

  || for tk_union
  {define {discriminator_type}:TypeCode
    || if indirect, locate ancestor and invoke method on it
    {if {= self._kind.val TK_indirect}
{return {{self.dereference}.discriminator_type}}}
    {if {not {= {self.kind}.val TK_union}}
{throw {new BadKindEx}}}
    || not worried about indirects here
    {return self.params.discriminator_type}}


  || for tk_union
  {define {default_index}:long
    || if indirect, locate ancestor and invoke method on it
    {if {= self._kind.val TK_indirect}
{return {{self.dereference}.default_index}}}
    {if {not {= {self.kind}.val TK_union}}
{throw {new BadKindEx}}}
    {if {void? self.params}
{throw {new InitializationEx}}}
    {println "typecode:default_index: checks done..."}
    {let mems:UnionMemberSeq=self.params.members
         count:ulong={self.params.members.length}
 i:ulong=0
 || default member is indicated by a zero octet label
 {while {< i count}
   {println "typecode:default_index: loop top..."}
   {let mem:UnionMember={mems.buff.aref i}
     || the zero octet marks the default index
     {println mem.label}
     {mem.label.item_type.val.print}
     {println "tc default_index: entering problme area..."}
     {if {and {mem.label.item_type.val.equal {TC_octet}}
```

```
      {= mem.label.item.val 0}}
 {return i}}}
   {set i {+ i 1}}}
 || if we get out of loop there is no default member
 {return -1}}}

  || for tk_string, tk_sequence, and tk_array
  {define {length}:ulong
    || if indirect, locate ancestor and invoke method on it
    {if {= self._kind.val TK_indirect}
{return {{self.dereference}.length}}}
    {if {not {or {= {self.kind}.val TK_sequence}
 {= {self.kind}.val TK_string}
 {= {self.kind}.val TK_array}}}
{throw {new BadKindEx}}}
    {if {void? self.params}
{throw {new InitializationEx}}}
    {return self.params.length}}


  || for tk_sequence, tk_array, tk_alias
  {define {content_type}:TypeCode
    || if indirect, locate ancestor and invoke method on it
    {if {= self._kind.val TK_indirect}
{return {{self.dereference}.content_type}}}
    {if {not {or {= {self.kind}.val TK_sequence}
 {= {self.kind}.val TK_alias}
 {= {self.kind}.val TK_array}}}
{throw {new BadKindEx}}}
    {if {void? self.params}
{throw {new InitializationEx}}}
    || return content type
    {return self.params.content_type}}

  || for TK_indirect, returns TypeCode indirect points to
  {define {dereference}:TypeCode
    {if {not {= self._kind.val TK_indirect}}
{throw {new BadKindEx}}}
    {let levels:ulong=self.params.offset
 i:ulong=0
         tc:TypeCode=self.parent
      {while {< i levels}
 {set tc tc.parent}
 {set i {+ i 1}}}
      {return tc}}}

  {define {offset}:TypeCode
    {if {not {= self._kind.val TK_indirect}}
{throw {new BadKindEx}}}
    {if {void? self.params}
{throw {new InitializationEx}}}
    {return self.params.offset}}

}
```

```
|| every kind of TypeCode gets its own parameter class which contains
|| all the additional data required to fully define the type

{define-class TkObjRefParams {}
  id: RepositoryId
  name: Identifier

  {define {init id:RepositoryId intName:Identifier}
    {set self.id {id.duplicate}}
    {set self.name {intName.duplicate}}}

  {define {equal? other:TkObjRefParams}:bool
    {return {self.id.equal? other.id}}}

  {define {copy parent:TypeCode}:TkObjRefParams
    {return {new TkObjRefParams self.id self.name}}}
}

{define-class TkSequenceParams {}
  content_type:TypeCode
  length:ulong

  {define {init memType:TypeCode maxlen:ulong}
    {set self.content_type {memType.copy}}
    {set self.length maxlen}}

  {define {equal? other:TkSequenceParams}:bool
    {return {and {= self.length other.length}
 {self.content_type.equal other.content_type}}}}

  {define {print}
    {printout "maxlen: "}
    {println self.length}
    {printout "type: "}
    {self.content_type.print}}

  {define {copy parent:TypeCode}:TkSequenceParams
    {let params:TkSequenceParams={new TkSequenceParams
      self.content_type self.length}
      {set params.content_type.parent parent}
      {return params}}}
}

{define-class TkStringParams {}
  length:ulong

  {define {init maxlen:ulong}
    {set self.length maxlen}}

  {define {equal? other:TkStringParams}:bool
    {return {= self.length other.length}}}
```

```
  {define {print}
    {printout "length: "}
    {println self.length}}

  {define {copy parent:TypeCode}:TkStringParams
    {return {new TkStringParams self.length}}}
}

{define-class TkArrayParams {}
  content_type:TypeCode
  length:ulong

  {define {init memType:TypeCode length:ulong}
    {set self.content_type {memType.copy}}
    {set self.length length}}

  {define {equal? other:TkArrayParams}:bool
    {return {and {= self.length other.length}
 {self.content_type.equal other.content_type}}}}

  {define {print}
    {printout "length: "}
    {println self.length}
    {printout "type: "}
    {self.content_type.print}}

  {define {copy parent:TypeCode}:TkArrayParams
    {let params:TkArrayParams={new TkArrayParams
   self.content_type self.length}
      {set params.content_type.parent parent}
      {return params}}}
}

{define-class TkAliasParams {}
  name: Identifier
  id: RepositoryId
  content_type: TypeCode

  {define {print}
    {printout "name: "}
    {println {self.name.toText}}
    {printout "id: "}
    {println {self.id.toText}}
    {printout "type: "}
    {self.content_type.print}}

  {define {init name:Identifier
id:RepositoryId
aliasType:TypeCode}
    {set self.name {name.duplicate}}
    {set self.id {id.duplicate}}
    {set self.content_type {aliasType.copy}}}

  {define {equal? other:TkAliasParams}:bool
```

```
    {return {self.content_type.equal other.content_type}}}

  {define {copy parent:TypeCode}:TkAliasParams
    {let params:TkAliasParams={new TkAliasParams
   self.name self.id self.content_type}
      {set params.content_type.parent parent}
      {return params}}}

}

{define-class TkStructParams {}
  id:RepositoryId
  name:Identifier
  members:StructMemberSeq

  {define {init name:Identifier id:RepositoryId members:StructMember-
Seq}
    {set self.name {name.duplicate}}
    {set self.id {id.duplicate}}
    {set self.members {members.copy}}}

  {define {equal? other:TkStructParams}:bool
    {if {not {and {self.id.equal? other.id}
  {= {self.members.length} {other.members.length}}}}
{return false}}
    || check members for equality
    {let length:ulong={self.members.length}
         i:ulong=0
      {while {< i length}
 {if {not {{self.members.aref i}.equal? {other.members.aref i}}}
     {return false}}
 {set i {+ i 1}}}}
    {return true}}

  {define {add-member name:String memType:TypeCode}:void
    {self.members.put {new StructMember name memType}}}

  {define {print}
    {printout "name: "}
    {self.name.print}
    {printout " id: "}
    {self.id.print}
    {printout " memCount: "}
    {println {self.members.length}}
    {println "members:"}
    {let length:ulong={self.members.length}
         i:ulong=0
      {while {< i length}
 {println "----------"}
 {{self.members.aref i}.print}
 {set i {+ i 1}}}}
    {println "----------------------"}}

  {define {copy parent:TypeCode}:TkStructParams
```

```
    {let params:TkStructParams={new TkStructParams
    self.name self.id self.members}
         count:ulong={params.members.length}
 i:ulong=0
      || fix parent pointers
      {while {< i count}
{let mem:StructMember={params.members.aref i}
  {set mem.type.parent parent}
  {set i {+ i 1}}}}
      {return params}}}

}

{define-class TkEnumParams {}
  id:RepositoryId
  name:Identifier
  members:EnumMemberSeq

  {define {init name:Identifier id:RepositoryId members:EnumMemberSeq}
    {set self.name {name.duplicate}}
    {set self.id {id.duplicate}}
    {set self.members {members.copy}}}

  {define {equal? other:TkEnumParams}:bool
    {if {not {and {self.id.equal? other.id}
  {= {self.members.length} {other.members.length}}}}
{return false}}
    {return true}}

  {define {add-member name:String}:void
    {self.members.put {name.duplicate}}}

  {define {print}
    {printout "name: "}
    {self.name.print}
    {printout " id: "}
    {self.id.print}
    {printout " memCount: "}
    {println {self.members.length}}
    {println "members:"}
    {let length:ulong={self.members.length}
         i:ulong=0
      {while {< i length}
 {println "----------"}
 {printout "name: "}
 {{self.members.aref i}.print}
 {println ""}
 {set i {+ i 1}}}}
    {println "----------------------"}}

  {define {copy parent:TypeCode}:TkEnumParams
    {return {new TkEnumParams self.name self.id self.members}}}

}
```

```
{define-class TkUnionParams {}
  id:RepositoryId
  name:Identifier
  members:UnionMemberSeq
  discriminator_type:TypeCode

  {define {init name:Identifier id:RepositoryId members:UnionMemberSeq
disctype:TypeCode}
    {set self.name {name.duplicate}}
    {set self.id {id.duplicate}}
    {set self.discriminator_type {disctype.copy}}
    {set self.members {members.copy}}}

  {define {equal? other:TkUnionParams}:bool
    {if {not {and {self.id.equal? other.id}
  {= {self.members.length} {other.members.length}}}}
{return false}}
    || check members for equality
    {let length:ulong={self.members.length}
         i:ulong=0
      {while {< i length}
 {if {not {{self.members.aref i}.equal? {other.members.aref i}}}
     {return false}}
 {set i {+ i 1}}}}
    {return true}}

  {define {add-member name:String memType:TypeCode
label:CORBA_Any}:void
    {self.members.put {new UnionMember name memType label}}}

  {define {print}
    {printout "name: "}
    {self.name.print}
    {printout " id: "}
    {self.id.print}
    {printout " memCount: "}
    {println {self.members.length}}
    {println "members:"}
    {let length:ulong={self.members.length}
         i:ulong=0
      {while {< i length}
 {println "----------"}
 {{self.members.aref i}.print}
 {set i {+ i 1}}}}
    {println "----------------------"}}

  {define {copy parent:TypeCode}:TkUnionParams
    {let params:TkUnionParams={new TkUnionParams self.name self.id
   self.members self.discriminator_type}
         count:ulong={params.members.length}
 i:ulong=0
      || fix parent pointers
      {set params.discriminator_type.parent parent}
```

```
      {while {< i count}
{let mem:StructMember={params.members.aref i}
  {set mem.type.parent parent}
  {set i {+ i 1}}}}
      {return params}}}


}

{define-constant TkExceptParams:type=TkStructParams}

{define-class TkIndirectParams {}
  offset:ulong

  {define {init offset:ulong}
    {set self.offset offset}}

  {define {equal? other:TkIndirectParams}:bool
    {return {= self.offset other.offset}}}

  {define {print}
    {printout "indirect: "}
    {println self.offset}}

  {define {copy parent:TypeCode}:TkIndirectParams
    {return {new TkIndirectParams self.offset}}}
}



{define {TC_null}:TypeCode
  {return {new TypeCode kind=TK_null}}}

{define {TC_void}:TypeCode
  {return {new TypeCode kind=TK_void}}}

{define {TC_short}:TypeCode
  {return {new TypeCode kind=TK_short}}}

{define {TC_long}:TypeCode
  {return {new TypeCode kind=TK_long}}}

{define {TC_ushort}:TypeCode
  {return {new TypeCode kind=TK_ushort}}}

{define {TC_ulong}:TypeCode
  {return {new TypeCode kind=TK_ulong}}}

{define {TC_float}:TypeCode
  {return {new TypeCode kind=TK_float}}}

{define {TC_double}:TypeCode
  {return {new TypeCode kind=TK_double}}}
```

```
{define {TC_boolean}:TypeCode
  {return {new TypeCode kind=TK_boolean}}}

{define {TC_char}:TypeCode
  {return {new TypeCode kind=TK_char}}}

{define {TC_octet}:TypeCode
  {return {new TypeCode kind=TK_octet}}}

{define {TC_any}:TypeCode
  {return {new TypeCode kind=TK_any}}}

{define {TC_TypeCode}:TypeCode
  {return {new TypeCode kind=TK_TypeCode}}}

{define {TC_Principal}:TypeCode
  {return {new TypeCode kind=TK_Principal}}}

{define {TC_string}:TypeCode
  {return {new TypeCode kind=TK_string params={new TkStringParams
0}}}}

{define {TC_sequence_octet}:TypeCode
  {return {create_sequence_tc 0 {TC_octet}}}}

|| any
{define-class CORBA_Any {}
  item:any
  item_type:TypeCodeP

  {define {init val:any=void theType:TypeCode={TC_void}}
    {printout "CORBA_ANy: init: val: "}
    {println val}
    {set self.item val}
    {printout "CORBA_Any: init: valtype: "}
    {theType.print}
    {set self.item_type {new TypeCodeP val=theType}}}

}
```

A.8 ORB/GIOP/giop.curl

```
|| File: ORB/GIOP/giop.curl
|| Defines: All GIOP messages.

{include "cdr.curl"}
{include "nvlist.curl"}
{include "reqmgr.curl"}
{include "profile.curl"}
{include "util.curl"}
```

```
|| GIOP message types
{define-constant GIOP_REQUEST_MSG:char=0}
{define-constant GIOP_REPLY_MSG:char=1}
{define-constant GIOP_CANCELREQUEST_MSG:char=2}
{define-constant GIOP_LOCATEREQUEST_MSG:char=3}
{define-constant GIOP_LOCATEREPLY_MSG:char=4}
{define-constant GIOP_CLOSECONNECTION_MSG:char=5}
{define-constant GIOP_MESSAGEERROR_MSG:char=6}

|| locations of stuff in GIOP messages
{define-constant GIOP_MSGTYPE_LOC:ulong=7}
{define-constant GIOP_BYTEORDER_LOC:ulong=6}
{define-constant GIOP_HEADER_SIZE:ulong=12}
{define-constant GIOP_LENGTH_LOC:ulong=8}
{define-constant GIOP_MSGSTART_LOC:ulong=12} || first byte after the
header

|| right now we're request id monogamous -- since we're only
|| allowing one outstanding request per connection right now,
|| we only need one request id ever.
|| for debugging purposes we're going to make it something that's
|| easily recognizable in a CDR-encoding
{define-constant REQUEST_ID:ulong=#xAABBCCDD}


{define-class GIOP-profile {Profile}
  status:StatusP
  exc:ExceptionP
  result:NamedValue
  args:NVList
  repl-expected:bool

  {define {init reqmgr:CORBA-request-manager}
    {invoke-method CORBA-Profile 'init self reqmgr}}

  {define {open}}
  {define {close}}
  {define {send-msg msg:GIOP-message}}
  {define {receive-msg}:GIOP-message}

  {define {get-obj-key}:String}

  {define {InvokeBlockedRequest op:text
repl-expected:bool
params:NVList
result:NamedValue
context:Context
status:StatusP
exc:exceptionP}:Status
    || generate CDR-encoded request message
    {let msg:GIOP-request-message={new GIOP-request-message
       REQUEST_ID repl-expected
       {self.get-obj-key}
       {text->String op} params}
```

```
    || initialize everything
    {set self.status status}
    {set self.exc exc}
    {set self.result result}
    {set self.args params}
    {set self.repl-expected repl-expected}

    ||| spawn a receiver thread -- see CORBA 2.0 spec, 12.7.1
    ||| ("TCP/IP Connection usage") for justification
    {new-thread self.wait-for-response}

    ||| send CDR request message
    {self.send-msg msg}

    || wait for response then return
    {while {= self.status.val NO_RETURN}
|| one second seems like a lot to sleep for, but that's
|| the granularity the call supports, so...
{sleep 1}}

    || everything should be good to go now
    {return STATUS_OK}}}

 {define {InvokeNonBlockedRequest op:text
  repl-expected:bool
  params:NVList
  result:NamedValue
  context:Context
  status:StatusP
  exc:exceptionP}:Status
   || generate CDR-encoded request message
   {let msg:GIOP-request-message={new GIOP-request-message
      REQUEST_ID repl-expected
      {self.get-obj-key}
      {text->String op} params}
    || initialize everything
    {set self.status status}
    {set self.exc exc}
    {set self.result result}
    {set self.args params}
    {set self.repl-expected repl-expected}

    ||| spawn a receiver thread -- see CORBA 2.0 spec, 12.7.1
    ||| ("TCP/IP Connection usage") for justification
    {new-thread self.wait-for-response}

    ||| send CDR request message
    {self.send-msg msg}

    ||| exit -- connection will be closed by receiving thread
    {return STATUS_OK}}}

 || all of wait-for-response is enclosed in a try block to give
 || reasonable behaivor -- i.e. catch any exception and pass it
```

```
  || up to higher layers in a standardized, thread-safe fashion
  {define {wait-for-response}:void
    {try
     {let msg:GIOP-message={self.receive-msg} || read message from
connection
|| dispatch on message type -- we can only receive certain messages
{cond {{= type GIOP_REPLY_MSG}
       {self.handle_reply msg}}
      {{= type GIOP_LOCATEREPLY_MSG}
       || we never send locate requests,
       || so we never receive locate replies
       {return}
       }
      {{= type GIOP_CLOSECONNECTION_MSG}
       {set self.exc.val {new COMM_FAILURE minor=CONNECTION_CLOSED}}
       {set self.status.val SYS_EXC_STATUS}
       {return}}
      {{= type GIOP_MESSAGEERROR_MSG}
       {set self.exc.val {new COMM_FAILURE minor=ERROR_RECEIVED}}
       {set self.status.val SYS_EXC_STATUS}
       {return}}
      {else
       || we got a message whose message type we didn't recognize.
       || close connection and throw an exception to let
       || the user know there was a problem
       {set self.exc.val {new COMM_FAILURE minor=BAD_REPLY}}
       {set self.status.val SYS_EXC_STATUS}
       {return}}
      }}}
     {catch {exc:StandardException}
    {set self.exc.val exc}
    {set self.status.val SYS_EXC_STATUS}}
     {catch {exc:Exception}
    {set self.exc.val exc}
    {set self.status.val USER_EXC_STATUS}}
     {finally {set self.exc {new UNKNOWN}}
      {set self.status.val SYS_EXC_STATUS}}}
    }

  || exceptions thrown inside handle_reply should be caught by
  || wait-for-request
  {define {handle_reply msg:GIOP-reply-msg}:void
    || first check the reply status
    {let replstatus:ReplyStatusType={msg.get-reply-status}
      {cond {{= replstatus NO_EXCEPTION}
     || construct new NVList in the format expected by
     || reply message's get-results function
     {let results:NVList={new NVList}
          count:ulong={self.args.get_count}
  i:ulong=0

       || first element in list is the result object
       {results.append self.result}
       || go through args list and append out and inout params
```

```
      {while {< i count}
 || fetch namedValue and check its arg_modes flag
 || to see if it's an out or inout
 {let nv:NamedValue={self.args.item i}
   {if {or {bit-or nv.arg_modes ARG_OUT}
   {bit-or nv.arg_modes ARG_INOUT}}
       {results.append nv}}}
 {set i {+ i 1}}}

       || results list ready, now just call repl msg's get-results
       {msg.get-results results}

       || all done, we can set status flag appropriately and exit
       {set self.status.val STATUS_OK}
       {return}}}
    {{or {= replstatus USER_EXCEPTION} {= replstatus
SYSTEM_EXCEPTION}}
     || we don't know how to unmarshal user exceptions,
     || so we'll extract the CDR encoding of the exception
     || (it's the only thing in the message!) and return it
     || enclosed in an appropriate exception. this probably
     || violates a number of abstraction barriers, but it's
     || the best way i can think of to get the data to the
     || user without making her jump through hoops.
     || we already know how to unmarshal system exceptions.

     || CDR decode-exception knows how to do above
     {msg.decode-exception self.exc {TC_null} {msg.body-start}}

     || set status and exc appropriately
     {if {= replstatus USER_EXCEPTION}
 {set self.status.val USER_EXC_STATUS}
 {set self.status.val SYS_EXC_STATUS}}
     {return}}
    {{= replstatus LOCATION_FORWARD}
     || we should do some complicated stuff here to extract the
     || new IOR and resend the request, but for now, we're just
     || going to throw an exception...
     {throw {new INV_OBJREF}}}
    }}}

}

{define {create-GIOP-message msgType:octet buff:CDR-buffer}:void
  {init-CDR-buffer buff}
  || set magic field to "GIOP"
  {buff.put-text "GIOP"}
  || set IIOP version number 1.0
  {buff.encode-byte 1}
  {buff.encode-byte 0}
  || indicate the byte-order
  {buff.encode-byte self.endian}
  || indicate message type
  {buff.encode-byte msgType}
```

```
  || initially set length of message to zero
  {buff.encode-word 0}}

{define {make-GIOP-msg txt:text}:GIOP-msg
  {let msgtype:char={aref txt GIOP_MSGTYPE_LOC}
    {cond {{= msgtype GIOP_REQUEST_MSG}
   {let msg:GIOP-request-msg={new GIOP-request-msg}
     {msg.appendText txt}
     {return msg}}}
  {{= msgtype GIOP_REPLY_MSG}
   {let msg:GIOP-reply-msg={new GIOP-reply-msg}
     {msg.appendText txt}
     {return msg}}}
  {{= msgtype GIOP_CLOSECONNECTION_MSG}
   {let msg:GIOP-closeconnection-msg={new GIOP-closeconnection-msg}
     {msg.appendText txt}
     {return msg}}}
  {{= msgtype GIOP_MESSAGEERROR_MSG}
   {let msg:GIOP-messageerror-msg={new GIOP-messageerror-msg}
     {msg.appendText txt}
     {return msg}}}
  {else
   || we're not really concerned with any of the other messages
   || right now since we don't handle them
   {let msg:GIOP-message={new GIOP-msg}
     {msg.appendText txt}
     {return msg}}}}}}

{define-class GIOP-msg {CDR-buffer}

  {define {init}
    {invoke-method CDR-buffer 'init self}}

  {define {set-length-field len:word}:void
    {self.put-word-posn len GIOP_LENGTH_LOCATION}}

  {define {get-length-field}:word
    {return {self.get-word-posn GIOP_LENGTH_LOCATION}}}

  {define {increment-length inc:int}:void
    {let len:word={self.get-length-field}
      {self.set-length-field {+ len inc}}}}

  {define {get-msg-type}:octet
    {return {self.get-byte-posn GIOP_MSGTYPE_LOC}}}

  {define {get-byte-order}:octet
    {return {self.get-byte-posn GIOP_BYTEORDER_LOC}}}

}

{define {create-GIOP-request-message reqid:ulong repl_expected:bool
     obj_key:String operation:String
     args:NVList}:GIOP-request-message
```

```
  {let msg:GIOP-request-message={new GIOP-request-message}
       bytes:int=0
    {make-GIOP-message GIOP_REQUEST_MESSAGE msg}
    || for now no ServiceContextList
    {set bytes {+ bytes {msg.encode-word 0}}}
    || request_id
    {set bytes {+ bytes {msg.encode-word reqid}}}
    || response_expected
    {if repl_expected
{set bytes {+ bytes {msg.encode-byte 1}}}
{set bytes {+ bytes {msg.encode-byte 0}}}}
    || object key
    {set bytes {+ bytes {msg.encode-value {obj_key.toBuffer}
  {tc-usequence-octet}}}}
    || operation
    {set bytes {+ bytes {msg.encode-value operation {tc-ustring}}}}

    || for now, not going to worry about the principal

    || encode args
    {let count:int={args.get_count}
         i:int=0
      {while {< i count}
{let val:any={args.item i}.argument.item
     tc:TypeCode={args.item i}.argument.item_type
  {set bytes {+ bytes {msg.encode-value val tc}}}}
{set i {+ i 1}}}}

    || context object? should it be part of the args list or should it
    || be a separate parameter?

    || update message length
    {msg.set-length-field bytes}}}


{define-class GIOP-request-msg {GIOP-msg}

  reqid-start:ulong
  op-start:ulong
  principal-start:ulong
  body-start:ulong

  {define {init}
    {invoke-method GIOP-msg 'init self}
    {set self.reqid-start 0}
    {set self.op-start 0}
    {set self.principal-start 0}
    {set self.body-start 0}}

  || skip-context-list is a utility routine that returns the index
  || of the first byte in a reply message after the service context
list
  {define {skip-context-list}:ulong
```

```
    {let numCtxts:int={self.get-word-posn GIOP_MSGSTART_LOC}
         length:ulong=0
         buff:sequence={get-buffer}
 i:int=0
 bytes:ulong=0
      {while {< i numCtxts}
|| increment bytes by the size of the next context structure.
|| the SIZEOF_WORD accounts for the context_id.
|| in the get-word-posn call, the first SIZEOF_WORD
|| accounts for the num contexts info, and the second for the
|| length of the context_data octet sequence
{set bytes {+ bytes SIZEOF_WORD {self.get-word-posn
 {+ GIOP_MSGSTART_LOC
    SIZEOF_WORD
    bytes
    SIZEOF_WORD}}}}
{set i {+ i 1}}}
      {return {+ GIOP_MSGSTART SIZEOF_WORD bytes}}}


  || the find methods provide some information caching about exactly
  || where in the message different parts are located. no find methods
  || are provided for pieces that are constant offsets from one of the
  || pieces that does have a find method. the clear-find-info clears
the
  || cache, forcing the message to be reexamined for the location
info.
  || this is useful if the message is edited or altered somehow.
  {define {clear-find-info}:void
    {set self.reqid-start 0}
    {set self.op-start 0}
    {set self.principal-start 0}
    {set self.body-start 0}}

  {define {find-reqid-start}:ulong
    {if {> self.reqid-start 0}
{return self.reqid-start}}
    {let start:ulong={align SIZEOF_WORD {self.skip-context-list}}
      {set self.reqid-start start}
      {return start}}}

  {define {find-op-start}:ulong
    {if {> self.op-start 0}
{return {self.op-start}}}
    {let length-start:ulong={+ SIZEOF_BYTE SIZEOF_WORD {self.find-
reqid-start}}
         length:ulong={self.get-word-posn length-start}
 start:ulong={align SIZEOF_WORD {+ length-start SIZEOF_WORD length}}
      {set self.op-start start}
      {return start}}}

  {define {find-principal-start}:ulong
    {if {> self.principal-start 0}
{return {self.principal-start}}}
```

```
    {let op-start:ulong={self.find-reqid-start}
         length:ulong={self.get-word-posn op-start}
 start:ulong={align SIZEOF_WORD {+ op-start SIZEOF_WORD length}}
      {set self.principal-start start}
      {return start}}}

  {define {find-body-start}:ulong
    {if {> self.body-start 0}
{return self.body-start}}
    {let principal-start:ulong={self.find-principal-start}
         length:ulong={self.get-word-posn principal-start}
      {set self.body-start {+ principal-start SIZEOF_WORD length}}
      {return {+ principal-start SIZEOF_WORD length}}}}



  {define {get-request-id}:request-id
    {return {self.get-word-posn {self.find-reqid-start}}}}

  {define {get-response-expected}:bool
    {let start:ulong={+ SIZEOF_WORD {self.find-reqid-start}}
         val:octet={self.get-byte-posn start}
      {if {= val 1}
  {return true}
  {return false}}}}

  {define {get-obj-key}:sequence
    {let start:ulong={+ SIZEOF_BYTE SIZEOF_WORD {self.find-reqid-
start}}
         buff:sequence={get-buffer}
         bytes:ulong={self.decode-value buff {tc-usequence-octet}}
      {if {= self.op-start 0}
  {set self.op-start {+ start bytes}}}
      {return buff}}}

  {define {get-op}:String
    {let s:String={new String}
 start:ulong={self.find-op-start}
 bytes:ulong={self.decode-string s start}
      {if {= self.principal-start 0}
  {set self.principal-start {+ start bytes}}}
      {return s}}}

}

|| routine to take a string like what might be produced by reading
|| from a NetworkPort and making it into a reply message
{define {cast-to-GIOP-reply-message t:text}:GIOP-reply-message
  {let msg:GIOP-reply-message={new GIOP-reply-message}
    {msg.appendText t}
    {set msg.endian {msg.get-byte-order}}
    {return msg}}}

{define-constant ReplyStatusType:type=ulong}
```

```
{define-constant NO_EXCEPTION:ulong=0}
{define-constant USER_EXCEPTION:ulong=1}
{define-constant SYSTEM_EXCEPTION:ulong=2}
{define-constant LOCATION_FORWARD:ulong=3}

{define-class GIOP-reply-msg {GIOP-msg}

  reqid-start:ulong

  {define {init}
    {invoke-method GIOP-message 'init self}
    {set self.reqid-start 0}}

  || skip-context-list is a utility routine that returns the index
  || of the first byte in a reply message after the service context
list
  {define {skip-context-list}:ulong
    {let numCtxts:int={self.get-word-posn GIOP_MSGSTART_LOC}
         length:ulong=0
         buff:sequence={get-buffer}
 i:int=0
 bytes:ulong=0
      {while {< i numCtxts}
|| increment bytes by the size of the next context structure.
|| the SIZEOF_WORD accounts for the context_id.
|| in the get-word-posn call, the first SIZEOF_WORD
|| accounts for the num contexts info, and the second for the
|| length of the context_data octet sequence
{set bytes {+ bytes SIZEOF_WORD {self.get-word-posn
 {+ GIOP_MSGSTART_LOC
    SIZEOF_WORD
    bytes
    SIZEOF_WORD}}}}
{set i {+ i 1}}}
      {return {+ GIOP_MSGSTART SIZEOF_WORD bytes}}}}


  || the find methods provide some information caching about exactly
  || where in the message different parts are located. no find methods
  || are provided for pieces that are constant offsets from one of the
  || pieces that does have a find method. the clear-find-info clears
the
  || cache, forcing the message to be reexamined for the location
info.
  || this is useful if the message is edited or altered somehow.
  {define {clear-find-info}:void
    {set self.reqid-start 0}}

  {define {find-reqid-start}:ulong
    {if {> self.reqid-start 0}
{return self.reqid-start}}
    {let start:ulong={align SIZEOF_WORD {self.skip-context-list}}
      {set self.reqid-start start}
      {return start}}}
```

```
  {define {get-request-id}:request-id
    {return {self.get-word-posn {self.find-reqid-start}}}}

  {define {get-reply-status}:ReplyStatusType
    {let start:ulong={+ SIZEOF_WORD {align SIZEOF_WORD
   {self.find-reqid-start}}}
      {return {self.get-word-posn start}}}}

  || get-results expects an NVList whose first element is the
  || object where the result is to be left, and whose remaining
  || memebers are the out and inout parameters for the operation
  || in the order they appear in the operation's IDL declaration
  {define {get-results params:NVList}:void
    {let start:ulong={+ {* 2 SIZEOF_WORD} {self.find-reqid-start}}
         count:int={params.get-count}
 bytes:int=0
 i:int=0
      {while {< i count}
{let curr:NamedValue={params.item i}
  {set bytes {+ bytes {self.decode-value curr.argument.item
 curr.argument.item_type
 {+ start bytes}}}}
  {set i {+ i 1}}}}}}

  {define {body-start}:ulong
    {let reqidstart:ulong={self.find-reqid-start}
      {return {+ reqidstart {* 2 SIZEOF_WORD}}}}}}
}

|| CloseConnection and MessageError messages consist only of the
|| GIOP header identifying the message type, so we don't need
|| any extra functions to extract data from them.
{define-class GIOP-closeconnection-msg {GIOP-msg}
  {define {init}
    {invoke-method GIOP-msg `init self}}
}

{define-class GIOP-messageerror-msg {GIOP-msg}
  {define {init}
    {invoke-method GIOP-msg `init self}}
}
```

A.9 ORB/GIOP/iiop.curl

```
|| File: ORB/GIOP/iiop.curl
|| Defines: IIOP-profile

{include "giop.curl"}
{include "cdr.curl"}
{include "debug.curl"}
```

```
{define-constant IIOP_MAJORVER_LOC:int=8}
{define-constant IIOP_MINORVER_LOC:int=9}
{define-constant IIOP_HOSTLENGTH_LOC:int=12}
{define-constant IIOP_HOSTSTART_LOC:int=16}

{define-class Version {}
  major:char
  minor:char

  {define {init major:char minor:char}
    {set self.major major}
    {set self.minor minor}}

  {define {print}
    {let maj:int=self.major
 min:int=self.minor
      {printout maj}
      {printout "."}
      {printout min}}}
}

{define {create-iiop-profile v:Version host:String
     port:ushort objkey:String}:IIOP-profile
  {println "create-iiop-profile top"}
  {let profile:IIOP-profile={new IIOP-profile {new CORBA-request-man-
ager}}
       len:ulong=0
    {set len {+ len {profile.encode-byte v.major}}}
    {set len {+ len {profile.encode-byte v.minor}}}
    {set len {+ len {profile.encode-value host {tc-ustring}}}}
    {set len {+ len {profile.encode-halfword port}}}
    {set len {+ len {profile.encode-value {objkey.toBuffer}
       {tc-usequence-octet}}}}
    {println "create-iiop-profile bottom"}
    {return profile}}}

{define {make-iiop-profile data:String}:IIOP-profile
  {let profile:IIOP-profile={new IIOP-profile}
    {set profile.endian {data.get 0}}
    {profile.concatenate data}
    {return profile}}}

{define-class IIOP-profile {CDR-encapsulation GIOP-profile}
  server:NetworkPort
  open?:bool

  {define {init}
    {invoke-method CDR-encapsulation 'init self}
    {invoke-method GIOP-profile 'init self}}

  {define {print}
    {let ver:Version={self.get-version}
      {printout "Version: "}
      {ver.print}
```

```
      {println ""}}
    {printout "host: "}
    {{self.get-host}.write standard-output-port}
    {println ""}
    {printout "port: "}
    {println {self.get-port}}
    {println "object key:"}
    {print-string-ascii {self.get-obj-key}}

  {define {get-version}:Version
    {let major:char={self.get-byte-posn IIOP_MAJORVER_LOC}
         minor:char={self.get-byte-posn IIOP_MINORVER_LOC}
      {return {new Version major minor}}}}

  {define {get-host}:String
    {let length:int={self.get-word-posn IIOP_HOSTLENGTH_LOC}
      || last character is null character. strip it out...
      {return {self.substring IIOP_HOSTSTART_LOC {- length 1}}}}}

  {define {get-port}:int
    {let host-length:int={self.get-word-posn IIOP_HOSTLENGTH_LOC}
      {return {self.get-halfword-posn {align SIZEOF_HALFWORD
     {+ IIOP_HOSTSTART_LOC
host-length}}}}}}

  {define {get-obj-key}:String
    {let host-length:int={self.get-word-posn IIOP_HOSTLENGTH_LOC}
         || finding the object key length is a bit complicated with
 || all the alignments that have to be done. A halfword (the port)
 || comes after the hostname, so we must align on a halfword boundary
 || following the hostname, then add in the halfword length
 || and align on a full word boundary.
         key-length-pos:int={align SIZEOF_WORD {+ SIZEOF_HALFWORD
  {align SIZEOF_HALFWORD
 {+ IIOP_HOSTSTART_LOC
    host-length}}}}
         key-length:int={self.get-word-posn key-length-pos}
      {return {self.substring {+ key-length-pos SIZEOF_WORD} key-
length}}}}

  {define {open}
    {if self.open?
{return}}
    {set self.server {connect-to-host {{self.get-host}.toText}
      {self.get-port}}}
    {set self.open? true}}

  {define {close}
    {self.server.Close}
    {set self.open? false}}

  {define {send-msg msg:GIOP-message}
    {if {not self.open?}
{self.open}}
```

```
    {msg.write self.server}
    {self.server.Flush}}

  {define {receive-msg}:GIOP-message
    || first read in the GIOP header and decide what to do from there
    {let hdr:text={new text GIOP_HEADER_SIZE}
         bytes:ulong={self.server.ReadText header GIOP_HEADER_SIZE}
      || if we couldn't read from port, throw a comm failure...
      {if {< bytes GIOP_HEADER_SIZE}
  {throw {new COMM_ERROR}}}
      || otherwise find size of message and read in rest of it,
      || then create an appropriate message based on the message type
      {let msg:GIOP-msg={make-GIOP-msg hdr}
   size:ulong={msg.get-length-field}
   type:byte={msg.get-msg-type}
   body:text={new text size}

|| read in remainder of message body
{set bytes {self.server.ReadText body size}}
|| if we couldn't read from port, throw a comm failure...
        {if {< bytes size}
    {throw {new COMM_ERROR}}}

|| attach body to header
{msg.appendText body}
{return msg}}}}
}
```

A.10 ORB/GIOP/cdr.curl

```
|| File: ORB/GIOP/cdr.curl
|| Defines: CDR-buffer class.

{include "typecode.curl"}
{include "debug.curl"}
{include "util.curl"}
{include "types.curl"}
{include "exceptions.curl"}

{define-constant SIZEOF_BYTE:ulong=1}
{define-constant SIZEOF_HALFWORD:ulong=2}
{define-constant SIZEOF_WORD:ulong=4}
{define-constant SIZEOF_DOUBLEWORD:ulong=8}

{define-constant LITTLE_ENDIAN:byte=1}
{define-constant BIG_ENDIAN:byte=0}

{define-constant PAD_VALUE:byte=#xaa}


|| utility to pad buff out to correct alignment depending on size
|| of data type
```

```
|| returns number of pads put in
{define {pad size:ulong buff:String}:ulong
  {if {= 0 {% {buff.len} size}}
      {return 0}}
  {let num:ulong={- size {% {buff.len} size}}
       num2:ulong=num
    {while {> num2 0}
   {buff.appendCh PAD_VALUE}
   {set num2 {- num2 1}}}
    {return num}}}

{define {align size:ulong index:ulong}:ulong
  {if {= 0 {% index size}}
      {return index}}
  {return {+ index {- size {% index size}}}}}

{define {init-CDR-buffer buff:CDR-buffer}
  {set buff.endian BIG_ENDIAN}}

{define {make-CDR-buffer s:String ordering:cahr}
  {let buff:CDR-buffer={new CDR-buffer}
    {set buff.endian ordering}
    {buff.concatenate s}
    {return buff}}}

|| a CDR-buffer is a buffer of properly aligned CDR-encoded data
|| it provides functions to encode all the kinds of CORBA types.
|| each function returns the number of bytes it writes into the
|| buffer (data + padding)
{define-class CDR-buffer {String}
  endian:char

  {define {init}
    {invoke-method String 'init self}
    {set self.endian LITTLE_ENDIAN}}

  {define {print}:void
    {let i:int=0
 c:word=0
        contents:text={self.toText}
 length:int={self.len}
 wordcount:int=0
      {println "------------------------------"}
      {while {< i length}
     {if {= 0 {% i 4}}
      {let
 {printout "-------- words: "}
 {println wordcount}
 {set wordcount {+ 1 wordcount}}}}
     {set c {aref contents i}}
     {println c}
     {set i {+ i 1}}}
      {println "------------------------------"}}}
```

```
  || all the put methods and encode methods return the number of bytes
  || they added to the buffer

  || utility to copy a byte over into a buffer at a specified location
  {define {put-byte-posn value:char posn:ulong}:ulong
    {self.set posn {% value 256}}
    {return SIZEOF_BYTE}}

  {define {get-byte-posn posn:ulong}:char
    {return {self.get posn}}}

  || utility to copy a halfword over into a buffer at a specified loca-
tion
  {define {put-halfword-posn value:word posn:int}:ulong
    {let val:word=value
i:int=0
        start:int=posn
increment:int=1
      {if {= self.endian BIG_ENDIAN}
  {begin {set increment -1}
 {set start {+ posn {- SIZEOF_HALFWORD 1}}}}}
      {while {< {abs i} SIZEOF_HALFWORD}
     {self.set {+ start i} {% val 256}}
     {set val {bit-srl val 8}}
     {set i {+ i increment}}}
      {return SIZEOF_HALFWORD}}}

  {define {get-halfword-posn posn:ulong}:word
    {let start:int=posn
 increment:int=1
         val:word=0
 i:int=0
      {if {= self.endian LITTLE_ENDIAN}
  {begin {set increment -1}
 {set start {+ posn {- SIZEOF_HALFWORD 1}}}}}
      {while {< {abs i} SIZEOF_HALFWORD}
     {set val {bit-sll val 8}}
     {set val {+ val {self.get {+ start i}}}}
     {set i {+ i increment}}}
      {return val}}}

  || utility to copy a word over into a buffer at a specified location
  {define {put-word-posn value:word posn:ulong}:ulong
    {let val:word=value
i:int=0
        start:int=posn
increment:int=1
      {if {= self.endian BIG_ENDIAN}
  {begin {set increment -1}
 {set start {+ posn {- SIZEOF_WORD 1}}}}}
      {while {< {abs i} SIZEOF_WORD}
     {self.set {+ start i} {% val 256}}
     {set val {bit-srl val 8}}
     {set i {+ i increment}}}
```

```
      {return SIZEOF_WORD}}}

  {define {get-word-posn posn:ulong}:word
    {let start:int=posn
 increment:int=1
         val:word=0
 i:int=0
      {if {= self.endian LITTLE_ENDIAN}
  {begin {set increment -1}
 {set start {+ posn {- SIZEOF_WORD 1}}}}}
      {while {< {abs i} SIZEOF_WORD}
     {set val {bit-sll val 8}}
     {set val {+ val {self.get {+ start i}}}}
     {set i {+ i increment}}}
      {return val}}}

  {define {put-text txt:text}:ulong
    {self.appendText txt}
    {return {length txt}}}

  {define {encode-byte value:byteP}:ulong
    || a byte requires no padding
    {self.appendCh value.val}
    {return SIZEOF_BYTE}}

  {define {decode-byte value:byteP posn:ulong}:ulong
    {set value.val {self.get-byte-posn posn}}
    {return SIZEOF_BYTE}}

  {define {encode-boolean value:boolP}:ulong
    {if value.val
{return {self.encode-byte {new byteP val=1}}}
{return {self.encode-byte {new byteP val=0}}}}}

  {define {decode-boolean value:boolP posn:ulong}:ulong
    {set value.val {self.get-byte-posn posn}}
    {return SIZEOF_BYTE}}

  {define {encode-halfword value:halfwordP}:ulong
    || make sure we are properly aligned
    {let pads:int={pad SIZEOF_HALFWORD self}
      || put the value into the message
      {self.put-halfword-posn value.val {self.len}}
      {return {+ pads SIZEOF_HALFWORD}}}}

  {define {decode-halfword value:halfwordP posn:ulong}:ulong
    {let aligned:int={align SIZEOF_HALFWORD posn}
      {set value.val {self.get-halfword-posn aligned}}
      {return {+ SIZEOF_HALFWORD {- aligned posn}}}}}

  {define {encode-word value:wordP}:ulong
    || make sure we are properly aligned
    {let pads:int={pad SIZEOF_WORD self}
      || put the value into the message
```

```
      {self.put-word-posn value.val {self.len}}
      {return {+ pads SIZEOF_WORD}}}}

  {define {decode-word value:wordP posn:ulong}:ulong
    {let aligned:int={align SIZEOF_WORD posn}
      {set value.val {self.get-word-posn aligned}}
      {return {+ SIZEOF_WORD {- aligned posn}}}}}

  {define {encode-struct value:any tc:TypeCode}:ulong
    {println "encode-struct: top..."}

    || a struct is encoded by encoding each of its members
    || in the order in which they're listed in the typecode
    {let count:int={tc.member_count}
         i:int=0
 bytes:ulong=0
      {printout "count: "}
      {println count}
      {while {< i count}
{printout "i: "}
{println i}
{let name:symbol={text->symbol {{tc.member_name i}.toText}}
  {println name}
  {let
      || get the value that corresponds to name
      val:any={get-slot-by-name {typeof value} name value}
      tcode:TypeCode={tc.member_type i}
      {printout "encode-struct: val: "}
      {println val}
      {set bytes {+ bytes {self.encode-value val tcode}}}}
  {set i {+ i 1}}}}
      {return bytes}}}

  {define {decode-struct value:any tc:TypeCode posn:int}:int
    {let count:int={tc.member_count}
         i:int=0
 bytes:ulong=0
 {printout "decode-struct: count: "}
 {println count}
      {while {< i count}
{let memtype:TypeCode={tc.member_type i}
     name:symbol={text->symbol {{tc.member_name i}.toText}}
     {printout "decode-struct: i: "}
     {println i}
  {set bytes {+ bytes {self.decode-value
       {get-slot-by-name {typeof value} name value}
       memtype
       {+ posn bytes}}}}
  {set i {+ i 1}}}}
      {return bytes}}}

  {define {encode-exception value:Exception tc:TypeCode}:ulong
    || an exception is encoded by encoding its repository id
    || followed by each of its members
```

```
    || in the order in which they're listed in the typecode
    {let count:int={tc.member_count}
         i:int=0
 bytes:ulong=0
      {set bytes {+ bytes {self.encode-string {tc.id} {TC_string}}}}
      {while {< i count}
{printout "i: "}
{println i}
{let name:symbol={text->symbol {{tc.member_name i}.toText}}
  {println name}
  {let
  || get the value that corresponds to name
  val:any={get-slot-by-name {typeof value} name value}
  tcode:TypeCode={tc.member_type i}
  {tcode.print}
  {set bytes {+ bytes {self.encode-value val tcode}}}}
{set i {+ i 1}}}
      {return bytes}}}}

  || decode-exception knows how to decode standard system
  || exceptions, without looking at tc. if tc is a TK_null
  || then decode-exception generates a USER_EXCEPTION exception
  || and stores the CDR-encoded exception inside it -- or more
  || accurately it stores the remainder of the buffer in the
  || exception on the assumption that the exception is the
  || only thing in the body of the buffer. if tc is any other
  || typecode kind, then it is used to decode the exception
  || into the exception instance stored at value.val
  {define {decode-exception value:ExceptionP tc:TypeCode
posn:ulong}:ulong
    || first check to see if encoded exception is a system
    || exception
    {let bytes:ulong={make-system-exception value self posn}
      {if {> bytes 0} || system exception was successfully decoded
  {return bytes}}}
    || otherwise we need to check typecode and take appropriate action
    {if {= {tc.kind}.val TK_null}
      || pull out rest of buffer and put it in a USER_EXCEPTION excep-
tion
      {let buff:CDR-buffer={make-CDR-buffer self.endian {self.sub-
string
 posn
 {- {self.len}
    posn}}}
|| create new exception and store it away
{set value.val {new USER_EXCEPTION buff=buff}}
{return {- {self.len} posn}}}}

    || otherwise it's just like decoding a struct, after we get the
    || id out...
    {let name:String={new String}
         bytes:ulong={self.decode-string name {TC_string} posn}
         count:int={tc.member_count}
         i:int=0
```

```
 bytes:ulong=0

      || store name in _id
      {set value.val._id name}
      || decode members
      {while {< i count}
{let memtype:TypeCode={tc.member_type i}
     name:symbol={text->symbol {{tc.member_name i}.toText}}
  {set bytes {+ bytes {self.decode-value
       {get-slot-by-name {typeof value} name value}
       memtype
       {+ posn bytes}}}}
  {set i {+ i 1}}}}
      {return bytes}}}

  {define {encode-union value:any tc:TypeCode}:ulong
    {println "encoding union..."}
    || Union tags are encoded as the discriminant tag followed by
    || the representation of the selected member
    {let disc:any=value._d
 discType:TypeCode={tc.discriminator_type}
 val:any={value._value}
 valType:TypeCode={tc.member_type {value._index}}
 bytes:ulong={self.encode-value disc discType}
      {return {+ bytes {self.encode-value val valType}}}}}

  {define {decode-union value:any tc:TypeCode posn:ulong}:ulong
    {let discType:TypeCode={tc.discriminator_type}
      {return 0}}}

  {define {encode-array value:any tc:TypeCode}:ulong
    {println "entering encode-array..."}
    {println {typeof value}}
    || arrays are encoded as their array elements in sequence
    {let length:ulong={tc.length}
         memType:TypeCode={tc.content_type}
         i:int=1
 bytes:ulong=0
      {println "encode-array: encoding array elements..."}
      {while {< i length}
     {printout "i: "}
     {println i}
     {aref value i}
     {println "aref done..."}
     {set bytes {+ bytes {self.encode-value {aref value i} memType}}}
     {set i {+ i 1}}}
      {return bytes}}}

  {define {decode-array value:any tc:TypeCode posn:ulong}:ulong
    {let memtype:TypeCode={tc.content_type}
         num:int={tc.length}
 len:ulong=0
 i:int=0
      {while {< i num}
```

```
{set len {+ len {self.decode-value {aref value i}
   memtype
   {+ posn len}}}}
{set i {+ i 1}}}
      {return len}}}

  {define {encode-sequence value:Sequence tc:TypeCode}:ulong
    || sequences are encoded as a length value followed by
    || the elements of the sequence. since all curl sequences
    || are unbounded, but CORBA expects some sequences to be
    || bound, a curl sequence that is longer than the maximum
    || length of its corresponding CORBA sequence type will be
    || truncated to the appropriate length.
    {let bound:int={tc.length}
         len:int={value.length}
 i:int=0
 memType:TypeCode={tc.content_type}
 bytes:ulong=0
      {if {and {!= bound 0} || we do not have an unbounded CORBA
sequence
      {> len bound}} || we have more than the max number of elems
  {set len bound}} || truncate sequence
      || encode length
      {set bytes {+ bytes {self.encode-word {new wordP val=len}}}}
      || encode elements of sequence
      {while {< i len}
     {let val:word={value.aref i}.val
       {printout "encode-sequence: val: "}
       {println val}}
     {set bytes {+ bytes {self.encode-value {value.aref i} memType}}}
     {set i {+ i 1}}}
      {return bytes}}}

||  {define {decode-sequence value:Sequence tc:TypeCode
posn:ulong}:ulong
||    {let memtype:TypeCode={tc.content_type}
||         aligned:ulong={align SIZEOF_WORD posn}
||         count:int={self.get-word-posn aligned}
|| i:int=0
|| bytes:ulong=0
|| ctype:type={value.content-type} || the type of data in the sequence
||      {while {< i count}
||{let newval:any={new ctype}
||  {set bytes {+ bytes {self.decode-value newval
|| memtype
|| {+ aligned
||    SIZEOF_WORD bytes}}}}
||  {value.put newval}}
||{set i {+ i 1}}}
||      {return {+ bytes SIZEOF_WORD {- aligned posn}}}}}

  {define {encode-string value:any tc:TypeCode}:ulong
    || strings are encoded as a length value followed by
    || the characters of the string. since all curl strings
```

```
    || are unbounded, but CORBA expects some strings to be
    || bound, a curl string that is longer than the maximum
    || length of its corresponding CORBA string type will be
    || truncated to the appropriate length.
    {let bound:int={tc.length}
         len:int={value.len}
 i:int=0
 contents:text={value.toText}
 bytes:ulong=0
     {if {and {!= bound 0} || we do not have an unbounded CORBA string
      {> len bound}} || we have more than the max number of chars
  {set len bound}} || truncate string
      || add 1 character for the null termination
      {set len {+ len 1}}
      || encode length
      {set bytes {+ bytes {self.encode-word {new wordP val=len}}}}
      || reset len to value we can use in a loop...
      {set len {- len 1}}
      || encode chars
      {while {< i len}
     {set bytes {+ bytes {self.encode-byte
  {new byteP val={aref contents i}}}}}
     {set i {+ i 1}}}
      || encode terminating null
      {set bytes {+ bytes {self.encode-byte {new byteP val=0}}}}
      {return bytes}}}

  || requires: posn to be the start of a string (not necessarily
aligned)
  || effects: returns index of the first element after the string
  {define {skip-string posn:ulong}:ulong
    {let start:ulong={align SIZEOF_WORD posn}
         length:ulong={self.get-word-posn start}
      {return {+ start SIZEOF_WORD length}}}}

  {define {decode-string val:String posn:ulong}:ulong
    {let aligned:ulong={align SIZEOF_WORD posn}
     length:ulong={- {self.get-word-posn aligned} 1} || subtract 1 to
remove
                                                     || trailing zero
      {val.concatenate {self.substring {+ aligned SIZEOF_WORD}
length}}
      || add byte to account for terminating 0
      {return {+ SIZEOF_BYTE length SIZEOF_WORD {- aligned posn}}}}}

  {define {encode-typecode tc:TypeCode}:ulong
    {return {self.do-encode-typecode tc {new LocStack}}}}

  {define {do-encode-typecode tc:TypeCode stack:LocStack}:ulong
    {stack.print}

      || handle indirect case specially
      {if {= tc._kind.val TK_indirect}
  {let
```

```
      {println "do-encode-tc: encoding indirect..."}
  {let tcstart:ulong={stack.value-at-depth {tc.offset}}
      bytes:ulong=0
    {printout "do-encode-tc: tcstart: "}
    {println tcstart}
    {printout "do-encode-tc: encoding indirect: length: "}
    {println {self.len}}
    {set bytes {+ bytes {self.encode-word
 {new wordP val=TK_indirect}}}}
    {printout "do-encode-tc: encoding indirect: bytes: "}
    {println bytes}
    {printout "do-encode-tc: encoding indirect: offset: "}
    {println {- tcstart {self.len}}}
    {set bytes {+ bytes {self.encode-word
{new wordP val={- tcstart {self.len}}}}}}

   {return bytes}}}
}
    {let kind:ulong={tc.kind}.val
 bytes:ulong=0
 pbytes:ulong=0 || bytes in parameter list...
 plenloc:ulong=0 || location of word giving length of param octet seq
 start:ulong={align SIZEOF_WORD {self.len}} || location of start
                                            || of typecode encoding

      || encode-typecode works by encoding necessary elements
      || on a case by case basis

      || for all types we encode kind first
      {set bytes {+ bytes {self.encode-word {new wordP val=kind}}}}

      || if type with empty param list then we're done so we can return
      {if {or {= kind TK_null}{= kind TK_void}{= kind TK_short}
       {= kind TK_ushort}{= kind TK_long}{= kind TK_ulong}
       {= kind TK_float}{= kind TK_double}{= kind TK_boolean}
       {= kind TK_char}{= kind TK_octet}{= kind TK_any}
       {= kind TK_TypeCode}{= kind TK_Principal}}
  {return bytes}}

      || string has a simple parameter list, so just encode it's
      || length and return
      {if {= kind TK_string}
{let
    {set bytes {+ bytes {self.encode-word
 {new wordP val={tc.length}}}}}
    {return bytes}}}

      || push location of current typecode onto stack
      {stack.push start}

      || we know now that we have a complex type, so leave space
      || for length of param string. record location of this word.
      {set plenloc {self.len}}
      {set bytes {+ bytes {self.encode-word {new wordP val=0}}}}
```

```
      || encode byte order for encapsulation -- parameter encoding
starts here
      {set pbytes {+ pbytes {self.encode-byte {new byteP
val=self.endian}}}}

      || encode id and name for applicable types
      {if {or {= kind TK_struct}{= kind TK_union}{= kind TK_alias}
      {= kind TK_except}{= kind TK_objref}{= kind TK_enum}}
       {let id:RepositoryId={tc.id}
    name:Identifier={tc.name}
 {set pbytes {+ pbytes {self.encode-string id {TC_string}}}}
 {set pbytes {+ pbytes {self.encode-string name {TC_string}}}}}}

      || encode disctype and default index for unions
      {if {= kind TK_union}
        {let
    {set pbytes {+ pbytes {self.do-encode-typecode
   {tc.discriminator_type} stack}}}
    {set pbytes {+ pbytes {self.encode-word
   {new wordP val={tc.default_index}}}}}}}

      || encode member count for applicable types
      {if {or {= kind TK_struct}{= kind TK_except}
      {= kind TK_enum}{= kind TK_union}}
        {let
    {set pbytes {+ pbytes {self.encode-word
   {new wordP val={tc.member_count}}}}}}}

      || encode content_type for sequences, arrays, and aliases
      {if {or {= kind TK_sequence}{= kind TK_array}{= kind TK_alias}}
{let
    {set pbytes {+ pbytes {self.do-encode-typecode
   {tc.content_type} stack}}}}}

      || encode length for sequences and arrays
      {if {or {= kind TK_sequence}{= kind TK_array}}
  {let
      {set pbytes {+ pbytes {self.encode-word
     {new wordP val={tc.length}}}}}}}

      || encode members for enum
      {if {= kind TK_enum}
  {let count:ulong={tc.member_count}
       i:ulong=0
    {while {< i count}
      {set pbytes {+ pbytes {self.encode-string
     {tc.member_name i}
     {TC_string}}}}
      {set i {+ i 1}}}}}

      || encode members for struct and exception
      {if {or {= kind TK_struct}{= kind TK_except}}
  {let count:ulong={tc.member_count}
       i:ulong=0
```

```
    {while {< i count}
      {set pbytes {+ pbytes {self.encode-string
     {tc.member_name i}
     {TC_string}}}}
      {set pbytes {+ pbytes {self.do-encode-typecode
     {tc.member_type i} stack}}}
      {set i {+ i 1}}}}}

      || encode members for union
      {if {= kind TK_union}
  {let count:ulong={tc.member_count}
       i:ulong=0
       default_index:long={tc.default_index}
       disctype:TypeCode={tc.discriminator_type}
    {while {< i count}
      {let label:CORBA_Any={tc.member_label i}
|| have to handle default case specially to make sure
|| enough bytes are encoded...
{if {= i default_index}
    {set pbytes {+ pbytes {self.encode-value
   {simple-zero disctype}
   disctype}}}
    {set pbytes {+ pbytes {self.encode-value
   label.item
   label.item_type.val}}}}
|| the rest is just like struct and except
{set pbytes {+ pbytes {self.encode-string
       {tc.member_name i}
       {TC_string}}}}
{set pbytes {+ pbytes {self.do-encode-typecode
       {tc.member_type i} stack}}}
{set i {+ i 1}}}}}}

      || now fill in length of param octet sequence
      {self.put-word-posn pbytes plenloc}

      || pop current typecode off stack
      {stack.pop}

      || return total bytes written
      {return {+ bytes pbytes}}}}

  || requires: posn to be the start of a typecode (not necessarily
aligned)
  || effects: returns index of the first element after the typecode
  {define {skip-typecode posn:ulong}:ulong
    {let start:ulong={align SIZEOF_WORD posn}
         kind:TCKind={self.get-word-posn start}
      {cond {{or {= kind TK_objref} {= kind TK_struct} {= kind
TK_union}
 {= kind TK_enum} {= kind TK_sequence} {= kind TK_array}
 {= kind TK_alias} {= kind TK_except}}
     || complex parameter list
     {let length:ulong={self.get-word-posn {+ start SIZEOF_WORD}}
```

```
      {return {+ start {* 2 SIZEOF_WORD} length}}}}
    {{or {= kind TK_string} {= kind TK_indirect}}
     || simple parameter list
     {return {+ start SIZEOF_WORD SIZEOF_WORD}}}
    {else
     || empty parameter list
     {return {+ start SIZEOF_WORD}}}}}}

  {define {decode-typecode tc:TypeCodeP posn:ulong}:ulong
      {return {self.do-decode-typecode tc posn {new LocStack}}}}

  || loc is the location where this typecode appears in the toplevel
typecode
  || it is used for calculating locations in indirects
  {define {do-decode-typecode tc:TypeCodeP posn:ulong stack:Loc-
Stack}:ulong
    {let kind:wordP={new wordP val=0}
 marker:ulong=posn
 tcstart:ulong=0
 bytes:ulong=0

      || decode-typecode works by decoding necessary elements
      || on a case by case basis

      || get kind first
      {set marker {+ marker {self.decode-word kind marker}}}

      {printout "do-decode-tc: top: kind: "}
      {println kind.val}

      || we need this to push the tc location onto the stack later
      {set tcstart {- marker SIZEOF_WORD}}

      {cond {{= kind.val TK_null}
     {set tc.val {TC_null}}
     {return {- marker posn}}}
    {{= kind.val TK_void}
     {set tc.val {TC_void}}
     {return {- marker posn}}}
    {{= kind.val TK_short}
     {set tc.val {TC_short}}
     {return {- marker posn}}}
    {{= kind.val TK_long}
     {set tc.val {TC_long}}
     {return {- marker posn}}}
    {{= kind.val TK_ushort}
     {set tc.val {TC_ushort}}
     {return {- marker posn}}}
    {{= kind.val TK_ulong}
     {set tc.val {TC_ulong}}
     {return {- marker posn}}}
    {{= kind.val TK_float}
     {set tc.val {TC_float}}
     {return {- marker posn}}}
```

```
  {{= kind.val TK_double}
   {set tc.val {TC_double}}
   {return {- marker posn}}}
  {{= kind.val TK_boolean}
   {set tc.val {TC_boolean}}
   {return {- marker posn}}}
  {{= kind.val TK_char}
   {set tc.val {TC_char}}
   {return {- marker posn}}}
  {{= kind.val TK_octet}
   {set tc.val {TC_octet}}
   {return {- marker posn}}}
  {{= kind.val TK_any}
   {set tc.val {TC_any}}
   {return {- marker posn}}}
  {{= kind.val TK_TypeCode}
   {set tc.val {TC_TypeCode}}
   {return {- marker posn}}}
  {{= kind.val TK_Principal}
   {set tc.val {TC_Principal}}
   {return {- marker posn}}}
  {{= kind.val TK_string}
   {let length:wordP={new wordP val=0}
     {set marker {+ marker {self.decode-word length marker}}}
     {set tc.val {create_string_tc length.val}}
     {return {- marker posn}}}}
  {{= kind.val TK_indirect}
   || this indirect case only handles space-conserving indirects,
   || not recursive ones. the recursive ones are handled in the
   || sequence specific code
   {let indirect:wordP={new wordP val=0}
        length:long={self.len}
loc:ulong=0
oldendian:char=self.endian
testword:word=0
     {set marker {+ marker {self.decode-word indirect marker}}}
     {set loc {+ length indirect.val}}
     || check byte-ordering at pointed-to byte
     {set testword {self.get-word-posn loc}}
     || we know all tckinds fit in the first byte, so we can
     || check the magnitude of the testword to see if we need
     || to change the endian setting
     {if {> testword 255}
 {if {= self.endian BIG_ENDIAN}
     {set self.endian LITTLE_ENDIAN}
     {set self.endian BIG_ENDIAN}}}
     || now we can decode typecode at indirect, knowing
     || that the byte-ordering will be correct
     || NOTE: the fact that we're changing the byte-ordering
     || variable here makes this part pf typecode NON-threadsafe.
     || in the future this should probably be wrapped in a lock
     || of some sort.
     {self.do-decode-typecode tc loc stack}
     || restore original endian
```

```
        {set self.endian oldendian}
        {return {- marker posn}}}}
    }

      || otherwise we're dealing with a type with a complex
      || parameter list
      {let plength:wordP={new wordP val=0}
   params:CDR-buffer=void
   pmarker:ulong=SIZEOF_WORD

|| push tcstart onto the stack
{stack.push tcstart}

|| get length and use it to extract params
{set marker {+ marker {self.decode-word plength marker}}}

{printout "do-decode-tc: plength found: marker: "}
{println marker}

{set params {extract-CDR-encapsulation
     {self.substring marker plength.val}}}

|| now decode params on case by case basis
{cond {{= kind.val TK_objref}
       {let id:RepositoryId={new String}
    name:Identifier={new String}
 {set pmarker {+ pmarker {params.decode-string id pmarker}}}
 {set pmarker {+ pmarker {params.decode-string name pmarker}}}
 {set tc.val {create_objref_tc id name}}
 {stack.pop}
 {return {- {+ marker pmarker} posn}}}}

      {{or {= kind.val TK_struct}{= kind.val TK_except}}
       {let id:RepositoryId={new String}
    name:Identifier={new String}
    members:StructMemberSeq={new StructMemberSeq}
    memcount:wordP={new wordP val=0}
    i:ulong=0
 || get id, name, and memcount
 {set pmarker {+ pmarker {params.decode-string id pmarker}}}
 {set pmarker {+ pmarker {params.decode-string name pmarker}}}
 {set pmarker {+ pmarker {params.decode-word memcount pmarker}}}
 || decode each member and store it in members
 {while {< i memcount.val}
   {let memname:Identifier={new String}
        tcp:TypeCodeP={new TypeCodeP val=void}
oldendian:char=self.endian
     {set pmarker {+ pmarker {params.decode-string memname
   pmarker}}}
      || to make sure byte-ordering is right for the
      || param list, we set self's endian to params,
      || then set it back when we're done
      {set self.endian params.endian}
      {set pmarker {+ pmarker {self.do-decode-typecode
```

```
      tcp
      {+ pmarker marker}
      stack}}}
     {set self.endian oldendian}
     {members.put {new StructMember memname tcp.val}}
     {set i {+ i 1}}}}}
 || create new typecode and return bytes eaten
 {if {= kind.val TK_struct}
     {set tc.val {create_struct_tc id name members}}
     {set tc.val {create_exception_tc id name members}}}
 {stack.pop}
 {return {- {+ marker pmarker} posn}}}}

      {{or {= kind.val TK_union}}
       {let id:RepositoryId={new String}
    name:Identifier={new String}
    dtcp:TypeCodeP={new TypeCodeP val=void}
    dindex:wordP={new wordP val=0}
    members:UnionMemberSeq={new UnionMemberSeq}
    memcount:wordP={new wordP val=0}
    i:long=0
 || get id, name, and memcount
 {set pmarker {+ pmarker {params.decode-string id pmarker}}}
 {set pmarker {+ pmarker {params.decode-string name pmarker}}}
 {set pmarker {+ pmarker {params.do-decode-typecode dtcp
    pmarker
    stack}}}
 {set pmarker {+ pmarker {params.decode-word dindex pmarker}}}
 {set pmarker {+ pmarker {params.decode-word memcount pmarker}}}
 || decode each member and store it in members
 {while {< i memcount.val}
   {let memname:Identifier={new String}
        tcp:TypeCodeP={new TypeCodeP val=void}
label:CORBA_Any=void
oldendian:char=self.endian
     || we have to handle the labels on a case-by-case basis
     {cond {{or {= {dtcp.val.kind} TK_long}
{= {dtcp.val.kind} TK_ulong}
{= {dtcp.val.kind} TK_enum}}
    {let val:wordP={new wordP val=0}
      {set pmarker {+ pmarker {params.decode-word
       val pmarker}}}
      {set label {new CORBA_Any val=val
      theType=dtcp.val}}}}
   {{or {= {dtcp.val.kind} TK_short}
{= {dtcp.val.kind} TK_ushort}}
    {let val:halfwordP={new halfwordP val=0}
      {set pmarker {+ pmarker {params.decode-halfword
       val pmarker}}}
      {set label {new CORBA_Any val=val
      theType=dtcp.val}}}}
   {{or {= {dtcp.val.kind} TK_char}}
    {let val:byteP={new byteP val=0}
      {set pmarker {+ pmarker {params.decode-byte
```

```
      val pmarker}}}
     {set label {new CORBA_Any val=val
     theType=dtcp.val}}}}
  {{or {= {dtcp.val.kind} TK_boolean}}
   {let val:boolP={new boolP}
     {set pmarker {+ pmarker {params.decode-boolean
      val pmarker}}}
     {set label {new CORBA_Any val=val
     theType=dtcp.val}}}}
  }

    || if this is the default member we need to
    || handle it specially
    {if {= i dindex.val}
      {set label {new CORBA_Any val={new byteP val=0}
      theType={TC_octet}}}}

    {set pmarker {+ pmarker {params.decode-string memname
  pmarker}}}
    || to make sure byte-ordering is right for the
    || param list, we set self's endian to params,
    || then set it back when we're done
    {set self.endian params.endian}
    {set pmarker {+ pmarker {self.do-decode-typecode
     tcp {+ pmarker marker} stack}}}
    {set self.endian oldendian}
    {members.put {new UnionMember memname tcp.val label}}
    {set i {+ i 1}}}}}
|| create new typecode and return bytes eaten
{set tc.val {create_union_tc id name dtcp.val members}}

{stack.pop}
{return {- {+ marker pmarker} posn}}}}

     {{or {= kind.val TK_enum}}
      {let id:RepositoryId={new String}
   name:Identifier={new String}
   members:EnumMemberSeq={new EnumMemberSeq}
   memcount:wordP={new wordP val=0}
   i:ulong=0
|| get id, name, and memcount
{set pmarker {+ pmarker {params.decode-string id pmarker}}}
{set pmarker {+ pmarker {params.decode-string name pmarker}}}
{set pmarker {+ pmarker {params.decode-word
 memcount pmarker}}}
|| decode each member and store it in members
{while {< i memcount.val}
  {let memname:Identifier={new String}
    {set pmarker {+ pmarker {params.decode-string memname
  pmarker}}}
    {members.put memname}
    {set i {+ i 1}}}}
|| create new typecode and return bytes eaten
{set tc.val {create_enum_tc id name members}}
```

```
{stack.pop}
{return {- {+ marker pmarker} posn}}}
     {{or {= kind.val TK_alias}}
      {let id:RepositoryId={new String}
   name:Identifier={new String}
   tcp:TypeCodeP={new TypeCodeP val=void}
   oldendian:char=self.endian
|| get content type
{set pmarker {+ pmarker {params.decode-string id pmarker}}}
{set pmarker {+ pmarker {params.decode-string name pmarker}}}
|| to make sure byte-ordering is right for the
|| param list, we set self's endian to params,
|| then set it back when we're done
{set self.endian params.endian}
{set pmarker {+ pmarker {self.do-decode-typecode tcp
 {+ pmarker
    marker}
 stack}}}
{set self.endian oldendian}
|| create new typecode and return bytes eaten
{set tc.val {create_alias_tc id name tcp.val}}
{stack.pop}
{return {- {+ marker pmarker} posn}}}}
     {{= kind.val TK_array}
      {let length:wordP={new wordP val=0}
   tcp:TypeCodeP={new TypeCodeP val=void}
   oldendian:char=self.endian
|| get content type
|| to make sure byte-ordering is right for the
|| param list, we set self's endian to params,
|| then set it back when we're done
{set self.endian params.endian}
{set pmarker {+ pmarker {self.do-decode-typecode tcp
 {+ pmarker
    marker}
 stack}}}
{set self.endian oldendian}
{set pmarker {+ pmarker {params.decode-word length pmarker}}}
|| create new typecode and return bytes eaten
{set tc.val {create_array_tc length.val tcp.val}}}
{stack.pop}
{return {- {+ marker pmarker} posn}}}
     {{= kind.val TK_sequence}
      || sequence differs from array in that it has to handle
      || recursive indirects
      {let length:wordP={new wordP val=0}
   tcp:TypeCodeP={new TypeCodeP val=void}

|| get content type, checking for an indirect
{let contentkind:wordP={new wordP val=0}
     eaten:ulong={params.decode-word contentkind pmarker}
     oldendian:char=self.endian
  {printout "do-decode-typecode: eaten: "}
  {println eaten}
```

```
   {printout "do-decode-typecode: checking contentkind: "}
   {println contentkind.val}
   || if content kind is an indirect then we need to
   || check if its recursive. if it is, then we figure
   || out the depth and use create_recursive_sequence_tc
   || to build the typecode. otherwise we just recurse
   || and let the indirect code up top catch it.
   {if {= contentkind.val TK_indirect}
     {let offset:wordP={new wordP val=0} || byte offset
  offsetloc:ulong={+ marker pmarker eaten}
  depth:ulong=0
       {println "do-decode-tc: decoding indirect..."}
       {println "do-decode-tc: decoding indirect: stack:"}
       {stack.print}
       || get indirect
       {set eaten {+ eaten {params.decode-word
    offset {+ pmarker eaten}}}}
       {printout "do-decode-tc: marker: "}
       {println marker}
       {printout "do-decode-tc: pmarker: "}
       {println pmarker}
       {printout "do-decode-tc: offsetloc: "}
       {println offsetloc}
       {printout "do-decode-tc: offset: "}
       {println offset.val}
       || find stack depth
       {set depth {stack.depth-of-value
   {+ offsetloc offset.val}}}
       {printout "do-decode-tc: depth: "}
       {println depth}
       {printout "do-decode-tc: loc: "}
       {println {+ offsetloc offset.val}}
       || if depth is -1, we know this isn't a recursive
       || indirect so we just exit the if and let the
       || normal sequence code handle it.
       {if {> depth -1}
 {let
   {set pmarker {+ pmarker eaten}}
   || get sequence length
   {set pmarker {+ pmarker {params.decode-word
    length pmarker}}}
   || create recursive typecode
   {set tc.val
{create_recursive_sequence_tc length.val
      depth}}
   || pop location stack
   {stack.pop}
   {return {- {+ pmarker marker} posn}}}}}

   || otherwise just decode content type and length
   || to make sure byte-ordering is right for the
   || param list, we set self's endian to params,
   || then set it back when we're done
   {set self.endian params.endian}
```

```
   {set pmarker {+ pmarker {self.do-decode-typecode tcp
    {+ pmarker
       marker}
    stack}}}
   {set self.endian oldendian}
   {printout "do-decode-tc: decoding normal sequence: pmarker: "}
   {println pmarker}
   {set pmarker {+ pmarker {params.decode-word
    length pmarker}}}

   || create new typecode and return bytes eaten
   {set tc.val {create_sequence_tc length.val tcp.val}}
   {stack.pop}
   {return {- {+ marker pmarker} posn}}}}}
       }
|| won't compile without this return
{return 0}}}}

  {define {decode-object-reference value:ObjectP posn:ulong}:ulong
    {let typeid:String={new String}
         bytes:ulong=0
      || first get the typeid
      {set bytes {+ bytes {self.decode-string typeid {TC_string}
posn}}}
      {set value.val {new Object typeid}}
      {let tpseq:TPSeq={new TPSeq}
{set bytes {+ bytes {self.decode-sequence tpseq
  {TC_sequence_TaggedProfile}
  {+ bytes posn}}}}
{set value.val.profiles tpseq}
{return bytes}}}}

  {define {encode-value value:any tc:TypeCode}:ulong
    {println "entering encode-value..."}
    {let kind:ulong={tc.kind}.val
      {printout "kind: "}
      {println kind}
      {cond {{or {= kind TK_short}
 {= kind TK_ushort}}
     {return {self.encode-halfword value}}}
    {{or {= kind TK_long}
 {= kind TK_ulong}
 {= kind TK_float}}
     {return {self.encode-word value}}}
    ||  {{= kind TK_double}
    ||   {return {self.encode-double value}}}
    {{or {= kind TK_char}
 {= kind TK_octet}}
     {return {self.encode-byte value}}}
    {{= kind TK_boolean}
     {println "encoding boolean..."}
     {return {self.encode-boolean value}}}
    {{= kind TK_TypeCode}
     {self.encode-typecode value}}
```

```
    {{= kind TK_struct}
     {return {self.encode-struct value tc}}}
    {{= kind TK_union}
     {return {self.encode-union value tc}}}
    {{= kind TK_enum}
     {return {self.encode-word value.val}}}
    {{= kind TK_array}
     {return {self.encode-array value tc}}}
    {{= kind TK_sequence}
     {return {self.encode-sequence value tc}}}
    {{= kind TK_string}
     {return {self.encode-string value tc}}}
    {{= kind TK_any}
     {let bytes:ulong={self.encode-typecode value.item_type}
       {set bytes {+ bytes {self.encode-value
    value.item value.item_type}}}
       {return bytes}}}
||  {{= kind TK_except}
||   {self.encode-exception value tc}}
    {{= kind TK_Principal}
     {return {self.encode-word {new wordP val=0}}}}
    {{= kind TK_alias}
     {return {self.encode-value value {tc.content_type}}}}
||  {{= kind TK_objref}
||   {self.encode-objref value}}
    {else {println "error: unrecognized type code..."}
  {return 0}}
    }}}

  {define {decode-value value:any tc:TypeCode posn:ulong}:ulong
    {println "entering decode-value..."}
    {let kind:ulong={tc.kind}.val
      {cond {{or {= kind TK_short}
 {= kind TK_ushort}}
     {return {self.decode-halfword value posn}}}
    {{or {= kind TK_long}
 {= kind TK_ulong}
 {= kind TK_float}
 {= kind TK_enum}}
   {return {self.decode-word value posn}}}
||  {{= kind TK_double}
||   {return {self.decode-double value posn}}}
  {{or {= kind TK_char}
       {= kind TK_octet}}
   {return {self.decode-byte value posn}}}
  {{= kind TK_boolean}
   {return {self.decode-boolean value posn}}}
  {{= kind TK_TypeCode}
   {println "decode-value: decoding typecode..."}
   {printout "decode-value: posn: "}
   {println posn}
   {return {self.decode-typecode value posn}}}
  {{= kind TK_struct}
   {return {self.decode-struct value tc posn}}}
```

```
  {{= kind TK_union}
   {return {self.decode-union value tc posn}}}
  {{= kind TK_array}
   {return {self.decode-array value tc posn}}}
   {{= kind TK_sequence}
   {return 0}}
   {{= kind TK_string}
   {return {self.decode-string value posn}}}
||  {{= kind TK_any}
||   {self.decode-typecode value.item_type posn}
||   {self.decode-value value.item value.item_type}}
||  {{= kind TK_except}
||   {self.decode-exception value tc}}
||  {{= kind TK_Principal}
||   {return {self.encode-word 0}}}
  {{= kind TK_alias}
   {return {self.decode-value value {tc.content_type} posn}}}
  {{= kind TK_objref}
   {self.decode-object-reference value posn}}
  {else {println "cdr.decode-value error: unrecognized type code..."}
{return 0}}
  }}}


}

{define-class CDR-encapsulation {CDR-buffer}

  {define {init}
    {invoke-method CDR-buffer 'init self}
    || encode byte-ordering
    {self.encode-byte {new byteP val=self.endian}}}

}

{define {extract-CDR-encapsulation s:String}:CDR-buffer
  {if {= {s.len} 0}
      {throw {new BadStringEx}}}
  {let buff:CDR-buffer={new CDR-buffer}
       endian:char={s.get 0}
     {set buff.endian endian}
     {buff.concatenate s}
     {return buff}}}

|| make-system-exception examines buffer to see if the exception con-
tained
|| therein is a system exception. if not then it returns 0 and leaves
|| exc unchanged. if it is a system exception then it is decoded into
|| a new SystemException object which is left at exc.val. the number
of
|| bytes consumed is returned.
{define {make-system-exception exc:ExceptionP buff:CDR-buffer
posn:ulong}:ulong
  {let name:String={new String}
```

```
       bytes:ulong={buff.decode-string name {TC_string} posn}

    || check to see if this is a system exception
    {if {not {system-exception? name}}
{return 0}}

    || if it is a system exception, then generate a new system excep-
tion
    || from the information encoded in the buffer
    {let minor:ulongP={new ulongP}
         status:completion_status={new ulongP}
      {set bytes {+ bytes {buff.decode-word minor {+ bytes posn}}}}
      {set bytes {+ bytes {buff.decode-word status {+ bytes posn}}}}

      || create new exception
      {set exc.val {new SystemException id=name status=status
minor=minor}}

      {return bytes}}}}

{define-class StackMem {}
  val:ulong
  next:StackMem

  {define {init val:ulong}
    {set self.val val}
    {set self.next void}}
}

{define-class LocStack {}
  stack:StackMem

  {define {init}
    {set self.stack void}}

  {define {push val:ulong}:void
    {printout "stack.push: val: "}
    {println val}
    {let mem:StackMem={new StackMem val}
      {set mem.next self.stack}
      {set self.stack mem}}}

  {define {pop}:ulong
    {if {void? self.stack}
{throw {new BoundsEx}}}
    {let val:StackMem=self.stack
      {set self.stack val.next}
      {set val.next void}
      {return val.val}}}

  {define {value-at-depth depth:ulong}:ulong
    {if {or {void? self.stack} {< depth 0}}
{throw {new BoundsEx}}}
    {let curr:StackMem=self.stack
```

```
 i:int=0
      {while {< i depth}
{printout "i: "}
{println i}
{if {void? curr.next}
    {throw {new BoundsEx}}}
{set curr curr.next}
{set i {+ i 1}}}
      {return curr.val}}}

  || depth-of-value returns -1 if val is not found on the stack
  {define {depth-of-value val:ulong}:long
    {if {void? self.stack}
{return -1}}
    {let i:ulong=0
 curr:StackMem=self.stack
      {while {not {void? curr}}
{if {= curr.val val}
    {return i}}
{set curr curr.next}
{set i {+ i 1}}}
      {return -1}}}

  {define {print}:void
    {let curr:StackMem=self.stack
depth:ulong=0
      {while {not {void? curr}}
 {printout "stack depth: "}
 {printout depth}
 {printout " value: "}
 {println curr.val}
 {set curr curr.next}
 {set depth {+ depth 1}}}}}}

}

|| simple zero produces a zero value for the simple type
|| described in tc
{define {simple-zero tc:TypeCode}:any
  {let kind:TCKind={tc.kind}
    {cond {{or {= kind TK_long}{= kind TK_ulong}{= kind TK_enum}}
   {return {new wordP val=0}}}
  {{or {= kind TK_short}{= kind TK_ushort}}
   {return {new halfwordP val=0}}}
  {{or {= kind TK_char}{= kind TK_octet}}
   {return {new byteP val=0}}}
  {{or {= kind TK_boolean}}
   {return {new boolP val=false}}}
  {else {return 0}}}}}
```

References

[1] *The Common Object Request Broker: Architecture and Specification*, Revision 2.0. Object Management Group, July 1995.

[2] Jon Siegel. `CORBA Fundamentals and Programming`. John Wiley & Sons, Inc., 1996.

[3] Steve Vinoski. "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments." 1996.

[4] Richard Mark Soley, ed. *Object Management Architecture Guide*. JohnWiley & Sons, Inc., 1995.

[5] Jeri Edwards, Dan Harkey, Robert Orfali. *The Essential Distributed Objects Survival Guide*. John Wiley & Sons, Inc., 1996.

[6] Thomas J. Mowbray, Ron Zahavi. *The Essential Corba: Systems Integration Using Distributed Objects*. John Wiley & Sons, Inc., 1995.

[7] Raphael C. Malveau, Thomas J. Mowbray. *CORBA Design Patterns*. John Wiley & Sons, Inc., 1997.

[8] Netscape Communications Corporation. *New Netscape ONE Platform Brings Distributed Objects To the Internet and Intranets*. July 29, 1996. Press release. URL: http://home.netscape.com/newsref/pr/newsrelease199.html.

5466-48