Bionic Buffalo Tech Note #40:

Initializing CORBA Applications

last revised Sunday 1 November 1998

INTRODUCTION

This Tech Note describes how to initialize C-language CORBA applications.

The initialization process consists of the following steps:

1. acquiring the CORBA Environment
2. initialization of the ORB
3. finding the initial services
4. obtaining initial object references
5. finding needed application objects

Each of these steps will be covered in more detail below. Except where noted, the CORBA specification standardizes the procedure, and it should be portable to compliant implementations.

In this document, the word Specification will refer collectively to the OMG documents which define CORBA and standard CORBA services.

ACQUIRING THE CORBA ENVIRONMENT

The CORBA Environment is a partially-opaque structure. The Environment is required as a parameter to almost all CORBA procedures, including procedures, which are defined by an object's Interface Definition Language (IDL) description.

The Specification defines the CORBA Environment as:

```
typedef struct CORBA_Environment
        {
        CORBA_exception_type        _major;
        ...
        }
        CORBA_Environment;
```

Upon return from a call to an object method, the ***_major*** element has one of the three values ***CORBA_NO_EXCEPTION***, ***CORBA_USER_EXCEPTION***, or ***CORBA_SYSTEM_EXCEPTION***. Other elements in the Environment structure are defined by the implementor, and are opaque to the user.

Although examples in the Specification declare the Environment as a static or global structure, there is no explicit guidance on how the Environment is to be initialized. Furthermore, static or global structures will not work in multithreaded applications, since a thread would expect its own unique value for ***_major***.

Bionic Buffalo's CORBA implementation provides a separate instance of the Environment for each thread. To acquire its own instance of the Environment, a thread must call ***eg_new_environment()***. (This procedure is Bionic Buffalo's proprietary solution to the problem of Environment acquisition and initialization.) The prototype is:

```
unsigned long       eg_environment_create
                        ( eg_environment_parms_t    * parms1,
                        CORBA_Environment           ** ev1 ) ;
```

The ***parms1*** parameter specifies details for threaded or multiprocessor environments. For almost all purposes, the ***parms1*** parameter can be specified as ***NULL***. The use of ***parms1*** is discussed elsewhere in the documentation.

eg_environment_create() uses the ***ev1*** parameter to return a pointer to the ***CORBA_Environment*** structure. This pointer is used subsequently in the following steps.

If successful, ***eg_environment_create()*** returns zero. Otherwise, it returns an error code indicating why the operation failed.

INITIALIZING THE ORB

The Specification defines an ORB initialization procedure as follows:

```
typedef char*       CORBA_ORBid;

extern CORBA_ORB    CORBA_ORB_init
                        ( int                   * argc,
                        char                    ** argv,
                        CORBA_ORBid             orb_identifier,
                        CORBA_Environment       * env );
```

Each implementation may define different values for these parameters. In general:

- The ***orb_identifier*** argument is used to select from among different possible ORBs. If the string is empty, then the default ORB is selected.

- ***argc*** and ***argv*** are used to pass additional arguments to the ORB. These are presumed to be taken from the command-line arguments to the application invoking ***CORBA_ORB_init()***.

- The Environment pointer ***env*** is taken from the previous step, ***eg_environment_create()***.

Bionic Buffalo's ORB will have a default behaviour when used with an empty ***orb_identifier*** string, and with no arguments ***(argc = 0*** and ***argv = NULL)*** passed to the ORB. The documentation explains other options, but they will not be needed by most applications.

The value returned by ***CORBA_ORB_init()*** is the object reference for the ORB itself. This object reference will be used in the next step.

FINDING INITIAL SERVICES

Each implementation will have an initial set of services available to the application. These are given character-string names. Certain names are standardized by the Specification. The standard service names are listed in the following table:

Name	Description
RootPOA	The root Portable Object Adapter (POA). (Object adapters are used to incorporate new objects into the system, and the POA is a standard type of object adapter described by the Specification.)
POACurrent	The current Portable Object Adapter (POA).
InterfaceRepository	The database of object and type interface definitions known to the system.
NameService	A directory service used for naming objects.
TradingService	A service used for advertising and discovering objects.
SecurityCurrent	A service to manage security in the current execution context.
TransactionCurrent	A service to manage transactions in the current execution context.

Each of these services is described in the Specification. Not all implementations will have all services available.

To obtain a list of services available initially with a given ORB, the application calls the ***CORBA_ORB_list_initial_services()*** procedure, which returns a list of strings. Each string is a service name chosen from the above table. The definitions are:

```
typedef char                    * CORBA_ORB_ObjectId ;

typedef struct
        {
        unsigned long           _maximum ;
        unsigned long           _length ;
        CORBA_ORB_ObjectId      * _buffer ;
        }
        CORBA_ORB_ObjectIdList ;

CORBA_ORB_ObjectIdList          * CORBA_ORB_list_initial_services
                                        ( CORBA_ORB             orb1,
                                        CORBA_Environment       * env1 ) ;
```

The next task for most applications will be to use one of the listed services to find the objects needed by the application. Two of these initial services are used for this purpose: *NameService* and *TradingService*.

The Name Service provides a tree-structured directory of objects, similar to a common file system directory. The Trading Service allows objects to advertise themselves with their properties, and permits potential clients to search for objects with specified qualities.

Regardless of the initial service needed, the name of the service (selected from the returned sequence of strings) is used in the next step.

OBTAINING INITIAL OBJECT REFERENCES

A name returned by ***CORBA_ORB_list_initial_services()*** may be submitted to another routine, ***CORBA_ORB_resolve_initial_references()***. This latter routine will return the object reference for the designated initial service. The function prototype is:

```
CORBA_Object   CORBA_ORB_resolve_initial_reference
                                ( CORBA_ORB             orb1,
                                CORBA_ORB_ObjectId      object_id1,
                                CORBA_Environment       * env1 ) ;
```

Each initial service must be submitted separately, since the routine resolves only a single reference at a time.

The reference to the service object may then be used to invoke the object's methods.

FINDING NEEDED APPLICATION OBJECTS

As mentioned above, there are two standard services used to find objects: the Name Service and the Trader Service.

The Name Service provides a tree-structured directory for objects. Each directory node is an object with the ***CosNaming_NamingContext*** interface. As with common file-system directories, there are operations to list the names in the directory, and to resolve the names to object references. There are also operations to bind objects to names in the context, or to rename objects by binding them to different names. The Specification does not provide a standard directory structure or layout.

The Trader Service matches servers and clients. Servers *export* descriptions of their objects to the Trader Service, and would-be clients *import* the descriptions from the Trader Service. A Trader Service client can qualify requests by using *properties*, so that a search for an object of a given type may be limited only to those objects with appropriate property values. Traders may communicate among themselves, allowing a single trader to provide access to objects originally exported to other traders.

Detailed discussions are reserved to two other documents:

- *Tech Note #41: The CORBA Name Service*
- *Tech Note #42: The CORBA Trader Service*

In addition, there is a third document specific to DSM-CC:

- *Tech Note #43: Initializing DSM-CC Applications*.

(DSM-CC specifies interfaces for objects used in multimedia systems, such as interactive television or video-on-demand. The ***DSM_Directory*** interface inherits the ***CosNaming_NamingContext*** interface.)