Bionic Buffalo Tech Note #52

Object Keys, Object Identifiers, and Adapter Names

last revised Tuesday 2003.07.29

Introduction

The CORBA specification defines data types and structures used to identify objects and adapters. However, while the types and structures are specified, the contents of some items are left to implementor decisions. This Tech Note discusses the conventions used by Bionic Buffalo's Tatanka™ implementations of CORBA software when creating object keys (used in interoperable object references, or IORs), object identifiers (OIDs), and portable object adapter (POA) names.

Except for application in `corbaloc:iiop` URLs, the content of these types and structures are transparent to most application software, but may be of interest to developers creating new servant or adapter implementations, or modifying existing code. For a discussion more relevant to application programmers, please refer to *Tech Note #100, Identifying CORBA Objects*.

These conventions, while compliant with the specification, are proprietary to Bionic Buffalo's software, and may not necessarily hold for software from other vendors. However, other developers are certainly welcome to adapt them to their own purposes.

Portable Object Adapter (POA) Names

The specification allows any non-NULL character to be used in a POA name. Bionic Buffalo ORBs permit any name allowed by the specification. However, Bionic Buffalo's own applications use a more restricted subset of characters. The subset is what are called *unreserved* characters in RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*. The unreserved characters consist of ASCII alphabetic characters (`A..Z` and `a..z`), digits (`0..9`), and what RFC 2396 calls *mark* characters:

```
- _ . ! ~ * ' ( )
```

If an application uses a POA name character from outside this subset, Tatanka™ ORBs will escape the character when it appears in object keys. Escaping is done using the `%xx` method described in RFC 2396, the method familiar with URLs, where `xx` are the two hexadecimal digits representing the value of the character being escaped. For example, a POA named `"nobody's POA"` will be represented as

`"nobody's%20POA"` in an object key.

The specification organizes POAs into a tree-structured hierarchy, beginning at the root POA. Tatanka™ software refers to individual POAs using a pathname similar to a filesystem path name, beginning with the root POA. For example, `"/alpha/beta"` refers to POA `beta`, which is a child of POA `alpha`, which is a child of the root POA. (The path name separator `"/"` is always escaped in the POA names themselves, so parsing is simplified.)

Object Identifier (OID) Names

Except for some pseudo objects and objects which are intrinsic to the ORB implementation, objects are implemented by servants, not by the ORB itself. Servants are separate application programs which implement objects. The ORB only brokers the communication among applications. A servant communicates with an ORB through an adapter. (Most adapters are portable object adapters, or POAs, whose interfaces are defined by the CORBA core specification.)

Within a given adapter, an object is uniquely identified by its object identifer (OID). The OID is opaque to the ORB. Object identifiers (OIDs) may be assigned by the system (ORB), or they may be assigned by the application or servant, depending on the POA's `IdAssignmentPolicy`. (When the servant allows the ORB to define the OIDs, it is simply the servant's use of a convenient service.)

A message to an object from a client application is passed to the POA by the ORB. The POA then passes the message to the servant, identifying the appropriate servant by using the OID.

The specification allows any octet to appear in an OID, including `NULL`s. The Tatanka™ ORBs also permit this, allowing an application or servant to use any value convenient to it.

Interoperable Object References (IORs) and Object Keys

In an interoperable object reference (IOR), an object is identified by the ORB's address and by the object's key. (The situation is slightly more complex than this, but this is a good approximation.) An IOR also contains the object's repository identifier.

> The repository identifier tells what type of object is referenced, but does not serve to identify the object directly. Of course, there may be many objects with the same repository identifier.
>
> The ORB's address is a network address, host name, pipe name, or other information used to communicate with the ORB. An IOR may contain more than one address, if the ORB is accessible in more than one way.
>
> The key provides a way for the ORB to identify the object. It may be defined arbitrarily by the ORB. However, as will be shown, it is efficient to assign in systematically as

explained below.

OIDs themselves cannot be used as object keys, since they do not uniquely identify objects. (Different applications or servants might use the same OIDs for different objects.) However, an OID must be unique for a given POA. Therefore, the combination of POA with OID makes a unique identifier within an ORB. An ORB might create an arbitrary key, and define some mapping from key to <*POA, OID*>, but most ORBs (including Bionic Buffalo's ORBs) reasonably take the more efficient approach of including the POA and OID in the key itself. Moreover, since a POA is uniquely identifiable by its path from the ORB's root POA, the POA pathname is usually used as the POA identifier.

(There is another problem with using a mapping from key to <*POA, OID*>. Object references may be persistent, and such persistent references must survive the ORB itself. Any mapping tables used would have to be communicated somehow among ORBs.)

The specification allows servants to create arbitrary OIDs, and applications may name POAs arbitrarily. However, there is no specified mapping between <*POA, OID*> and object key. What follows explains how the Tatanka™ ORBs implement the mapping.

An object key consists of a series of named values. Each named value is of the form

<*tag*>=<*value*>

The named values are separated using the `":"` (colon) character. Two standard tags are `poa` and `oid`, whose values are the object's POA pathname and object identifier, respectively. For example, a complete key might be

```
poa=ns/fs:oid=/home/ralph/somefile.txt
```

which specifies the *ns/fs* POA, and OID */home/ralph/somefile.txt*.

The POA value determined as explained above: all but the unreserved characters in each POA name are escaped, and the `"/"` (solidus or slash) separator is inserted between names in the POA path.

The OID value is computed from the internal value (given by the servant) by escaping all characters except the following:

the alphabetic characters: `A..Z` and `a..z`
the digits: `0..9`
the mark characters: `- _ . ! ~ * ' ( )`
other unescaped characters: `; / ,`

(This is the same set as allowed for POA names, with the addition of the semicolon, solidus, and comma.)

The following table summarizes the tags recognized by all Tatanka™ ORBs.

tag	*description of use*
`dbg`	debug information; interpretation is ORB-specific
`oid`	object identifier, as known to servant, but escaped as described above
`poa`	POA pathname (formed from POA names as described above)
`rem`	human-readable remarks, ignored by ORB
`src`	information about the source of the key (who and when created)
`ver`	version of this format (this is A.0.0.3)

Some Tatanka™ ORBs recognize other tags. Some also permit metadata, allowing indirect references, compression, variable substitution, and other features. Their use is ORB-dependent, and the interpretation of specific tags and metadata are not discussed here.

Use of Object Keys in `corbaloc:iiop` URLs

A `corbaloc:iiop` URL allows the specification of object keys. It is a string of the form

corbaloc:<address_list>/<object_key>

where each member of the *<address_list>* is an IIOP address in a specified form. The *<object_key>* is mapped from the object key used in the IOR. The mapping is done by escaping octets or characters which are not permitted in URLs. In Bionic Buffalo ORBs, such forbidden characters are already escaped (as described above), so the *<object_key>* in the `corbaloc:iiop` URL is the same as the object key used in the IOR.

More characters are permitted unescaped in URL object keys than are permitted in POA names or in OID values. The additional permitted characters are

`: ? @ & = + $`

These characters (but no others) may be used unescaped by specific ORBs for metadata or within additional named values. Bionic Buffalo ORBs will not create object keys from characters which require escaping.

When the syntax and semantics of an OID are understood, `corbaloc:iiop` URLs can be used to create object references without the use of naming or trading services. (This is normally the only situation where applications might need to know about object keys. Normally, object keys are opaque to applications.) For example, by passing the string

`“corbaloc:iiop:abc.def.com/poa=ns/fs:”`

```
"oid=/home/ralph/somefile.txt"
```

to the `CORBA_ORB_string_to_object()` function, an application can acquire a reference to the file */home/ralph/somefile.txt* on the machine *abc.def.com*. This capability is especially useful during system initialization, when an application may require object references for remote services. (The Naming Service and the Trading Service provide other ways to discover initial references.)
