The parameter to this procedure is:

`oid:` the identifier of the object.

On a successful return, an `isobject` structure describing the object is returned.

Chapter 10

Defining New Services

The OSI Directory is used to register new services. The steps involved are simple:

- the application context and abstract syntax for the service is registered in the *isobjects*(5) file, this allows for your program to use the textual designator for these values rather than the object identifier form; and,
- an entry is created in the Directory containing information indicating where the service resides in the network, and, optionally, what UNIX program will be invoked whenever there is an incoming connection for the service.

Please read the following important message.

> **NOTE:** Consult the *READ-ME* file for information on how to generate the initial Directory entries for your system.

10.1 Standard Services

The "standard" approach is used when an incoming connection should result in *tsapd* invoking the service.

First, edit the *isoentities*(5) database (described in Chapter 7), and add the appropriate lines to define

- the abstract syntax used by the service (the description of the data structures exchanged at the application layer); and,

- the application context providing the service (the description of the service elements contained in the application layer along with the interactions between them and the presentation layer).

These are object identifiers (as described in Section 5.4.6 on page 135). The object identifier tree `1.17.2` has been usurped for defining local services using *The ISO Development Environment*. Choose `n`, where `n` is the lowest unassigned number in the `1.17.2.n` subtree, for use by the service (`n` should be an integer greater than `0`). Now edit the *isobjects*(5) file thusly:

```
"local service pci"    1.17.2.n.1
"local service"        1.17.2.n.2
```

Second, add an entry to the Directory so that initiator entities can locate the service. The following attributes will have to be entered:

commonName: The name that the entry is registered under, (e.g., `cn=servicestore`).

presentationAddress: The location in the network where the service resides, consisting of a transport-selector and one or more network addresses (if your host is multi-homed). It is easiest to use a 16–bit binary space for the transport selector: choose `t`, where `t` is the lowest unassigned TSAP ID between `1024` and `2047` inclusive.

supportedApplicationContext: The application context providing the service. Use the same string that you entered in the *isobjects*(5) file.

execVector: The command string which *tsapd* will invoke for each incoming connection. The first token is interpreted relative to the ISODE `SBINDIR` directory (usually */usr/etc/*).

So, the entry looks something like this:

```
objectClass= top & applicationEntity & quipuObject
objectClass= iSODEApplicationEntity
cn= servicestore
presentationAddress= #t/Internet=mydomainame
supportedApplicationContext= local service
acl=
execVector= program arg1 arg2 ... argN
```

10.2 Static Servers

In Section 2.5, two different server disciplines, the dynamic and the static approaches, were described. Thus far, this chapter has described the dynamic approach. The distinguishing mechanism of this discipline depends on whether the `objectClass` attribute for the entry contains the value

```
iSODEApplicationEntity
```

If so, then the entry describes a dynamic approach, and *tsapd* will listen for incoming connections.

If not, then the entry does not contain a `execVector` attribute, and *tsapd* will ignore the entry in the Directory. In order to avoid listening conflicts with *tsapd*, the network address chosen should be different:

- If the network-address is TCP-based, use a TCP port, `p`, that is different than the one used by *tsapd* (which, by default, is 102), e.g.,

  ```
  presentationAddress= #t/Internet=mydomainame+p
  ```

- If the network-address is X.25–based, use an X.25 protocol-ID, `p`, that is different than the one used by *tsapd* (which, by default is empty), e.g.,

  ```
  presentationAddress= #t/Int-X25=mydomainame+PID+p
  ```

- If the network-address is CLNS–based, then simply choose any unused transport-selector, `p`, e.g.,

  ```
  presentationAddress= #t/NS+470005...
  ```

After making a suitable entry in the Directory, the server program must be started (either by hand or automatically from some system startup file, e.g., */etc/rc.local*) without any arguments. The server program, using the `isodeserver` routine described on page 44 will then perform the appropriate actions to start listening on the desired addresses.

Finally, note that in order for the `isodeserver` routine to work correctly, you will need an entry in the *isoaliases*(5) file mapping the local host name (as returned by `PLocalHostName`) into a Directory Distinguished Name, e.g.,

```
hubris    hubris, cs, university college london, gb
```

This is necessary so that the call to `_str2aei` (or `str2aeinfo`) can find the appropriate application-entity information which is subsequently passed to `isodeserver`.

Part V

Appendices

Appendix A

Old-Style Associations

As discussed in Section 2.1, there are several ways to establish an association. In the interests of compatibility with previously specified applications, the *librosap*(3n) and *librtsap*(3n) libraries support "old-style" mechanisms for binding associations. These are discussed in this appendix. Use of these facilities for new applications is strongly discouraged.

A.1 Remote Operations

Under old-style association handling for remote operations, the mechanism used for establishing the association determines the remote operations service discipline to be used. If the basic service discipline is desired, then continue reading this section which describes the addressing and initialization mechanisms for native ROS associations. These make use of the RO-BEGIN and RO-END primitives. Instead, if the advanced service discipline is desired, then read Section A.2 on page 175 which describes how associations are established and released by the reliable transfer service (RTS) using the X.410 OPEN and X.410 CLOSE primitives. (Be sure to also read Section 3.3.1 on page 62).

A.1.1 Addresses

Addresses at the remote operations service access point are represented by the `RoSAPaddr` structure.

```
struct RoSAPaddr {
    struct SSAPaddr roa_addr;

    u_short     roa_port;
};
```

This structure contains two elements:

`roa_addr`: the session address, as described in Section 3.2 on page 42 of *Volume Two*; and,

`roa_port`: the port number of the entity residing above the RoSAP.

In Figure A.1, an example of how one constructs the RoSAP address for the directory services entity on host `RemoteHost` is presented. (The routine `is2saddr` is described on page 42 of *Volume Two*.)

A.1.2 Association Establishment

The *librosap*(3n) library distinguishes between the user which started an association, the *initiator*, and the user which subsequently was bound to the association, the *responder*. We sometimes term these two entities the *client* and the *server*, respectively.

Server Initialization

The *tsapd*(8c) daemon, upon accepting a connection from an initiating host, consults the ISO services database to determine which program on the local system implements the desired RoSAP entity. In the case of the remote operations service, the *tsapd* program contains the bootstrap for the remote operations provider. The daemon will again consult the ISO services database to determine which program on the system implements the desired RoSAP entity.

Once the program has been ascertained, the daemon runs the program with any arguments listed in the database. In addition, it appends some *magic arguments* to the argument vector. Hence, the very first action performed by the responder is to re-capture the RoSAP state contained in the magic arguments. This is done by calling the routine `RoInit`, which on a successful return, is equivalent to a RO-BEGIN.INDICATION from the remote operations service provider.