Part IV

Databases

Chapter 6

The ISO Aliases Database

The database *isoaliases* in the ISODE `ETCDIR` directory (usually */usr/etc/*) contains a simple mapping between names (terse strings) and values (e.g., user-friendly names and distinguished names).

The database itself is an ordinary ASCII text file containing an entry for each locally defined alias. Each entry contains

- the alias, a simple string; and,
- a user-friendly name or a distinguished name.

Blanks and/or tab characters are used to seperate items. However, double-quotes may be used to prevent separation for items containing embedded whitspace. The sharp character ('`#`') at the beginning of a line indicates a commentary line.

6.1 Accessing the Database

The *libacsap*(3n) library contains the routines used to access the database. There is one high-level routine, `alias2name` which returns the value which corresponds to an alias in the database.

```
char   *alias2name (name)
char   *name;
```

The parameter to this procedure is:

`name:` the alias to lookup.

This returns the manifest constant `NULLCP` if the given alias is not in the database.

In order to load specific aliases other than those read in the *isoaliases*(5) file, use the routine `add_alias`:

```
int     add_alias (name, value)
char   *name,
       *value;
```

The parameters to this procedure are:

`name:` the alias to enter; and,

`value:` its value.

This returns the manifest constant `NOTOK` if the given alias cannot be added.

6.2 User-Specific Aliases

By default a user-specific aliases database is consulted before the system-wide aliases file. The user-specific file is called *$HOME/.isode_aliases* in the user's home directory.

Chapter 7

The ISODE Entities Database

The database *isoentities* in the ISODE `ETCDIR` directory (usually */usr/etc/*) contains a simple mapping between application-entity information and presentation addresses. This database is used by the stub-directory service.

> **NOTE:** Use of this database is deprecated. Consult Chapter 10 on page 160 for further information.

The database itself is an ordinary ASCII text file containing information regarding the known application-entities on the network. Each entry contains:

- the object descriptor of the application-entity information, realized as a designator-qualifier (host/service) pair (the distinguished designator `default` is used for a template entry);
- the object identifier of the application-entity information expressed in dot-notation (if no application-entity information is desired, the string "NULL" should be used instead); and,
- the presentation address expressed in the string format described in Section 2.2.1 on page 20 in *Volume One*.

  Note that since double-quotes are often used in the new string format, it is **very** important to quote them correctly in the *isoentities*(5) file. Usually preceeding the first character of the address with backslash ("\") is adequate.

Blanks and/or tab characters are used to seperate items. However, double-quotes may be used to prevent separation for items containing embedded whitspace. This is useful for defining the presetnation address. The sharp character ('`#`') at the beginning of a line indicates a commentary line.

It is suggested for readability purposes that a blank line should seperate entries.

RFC1085 Support

Since applications using RFC1085 (the lightweight presentation protocol) usually demultiplex on the basis of TCP or UDP port, a further definition for the qualifier is placed in */etc/services*, one of:

```
qualifier    portno/lpp
qualifier    portno/tcp
qualifier    portno/udp
```

The first alternative says that the service lives on both TCP and UDP; the second alternative says that the service lives on TCP only; and, the third alternative says that the service lives on UDP only.

7.1 Accessing the Database

The *libacsap*(3n) library contains the routines used to access the database. The high-level routines, `_str2aei`, which returns application-entity information, and `aei2addr`, which maps the application-entity information into a presentation address have already been discussed in Section 2.2.1. Similarly, the "old-style" routines, `is2paddr`, `is2saddr`, `is2taddr`, which maps a host and service directly into a prsentation, session, or transport address (respectively) are discussed in Sections 2.2, 3.2, and 4.1 of *Volume Two* (respectively). Hence, for the remainder of this chapter, we need only consider the internal form used by the library in querying the database.

The `isoentity` structure is the internal form.

```
struct isoentity {
    OIDentifier ie_identifier;
    char    *ie_descriptor;
```

```
    struct PSAPaddr ie_addr;
};
```

The elements of this structure are:

`ie_identifier:` an object identifier used to form application-entity information;

`ie_identifier:` the corresponding object descriptor; and,

`ie_addr:` the presentation address.

The routine `getisoentity` reads the next entry in the database, opening the database if necessary.

```
struct isoentity *getisoentity ()
```

It returns the manifest constant `NULL` on error or end-of-file.

The routine `setisoentity` opens and rewinds the database.

```
int     setisoentity (f)
int     f;
```

The parameter to this procedure is:

`f:` the "stayopen" indicator, if non-zero, then the database will remain open over subsequent calls to the library.

The routine `endisoentity` closes the database.

```
int     endisoentity ()
```

Both of these routines return non-zero on success and zero otherwise.

Chapter 8

The ISO Macros Database

The database *isomacros* in the ISODE `ETCDIR` directory (usually */usr/etc/*) contains a simple mapping between user-friendly strings and network addresses. This database is used by when trying to resolve textual respresentations of network addresses (as desccribed in Section 2.2.1) for use with the network.

The database itself is an ordinary ASCII text file containing an entry for each locally defined macro. Each entry contains

- the macro, a simple string; and,
- the prefix of the corresponding network address.

Blanks and/or tab characters are used to seperate items. However, double-quotes may be used to prevent separation for items containing embedded whitspace. The sharp character ('`#`') at the beginning of a line indicates a commentary line.

Unlike the other databases in the ISODE, the user may not directly access this file. The routine `str2paddr` and `paddr2str` use this automatically.

8.1 User-Specific Macros

By default a user-specific macros database is consulted before the system-wide macros file. The user-specific file is called *$HOME/.isode_macros* in the user's home directory.

Chapter 9

The ISODE Objects Database

The database *isobjects* in the ISODE `ETCDIR` directory (usually */usr/etc/*) contains a simple mapping between object descriptors and object identifiers.

The database itself is an ordinary ASCII text file containing information regarding the known objects on the host. Each line contains

- the descriptor of the object, a simple string; and,
- the corresponding object identifier.

Blanks and/or tab characters are used to separate items. However, double-quotes may be used to prevent separation for items containing embedded whitspace. The sharp character ('`#`') at the beginning of a line indicates a commentary line.

9.1 Accessing the Database

The *libpsap*(3) library contains the routines used to access the database. These routines ultimately manipulate an `isobject` structure, which is the internal form.

```
struct isobject {
    char          *io_descriptor;

    OIDentifier    io_identity;
};
```

The elements of this structure are:

`io_descriptor:` the object descriptor; and,

`io_identity:` the object identifier.

The routine `getisobject` reads the next entry in the database, opening the database if necessary.

```
struct isobject *getisobject ()
```

It returns the manifest constant `NULL` on error or end-of-file.

The routine `setisobject` opens and rewinds the database.

```
int     setisobject (f)
int     f;
```

The parameter to this procedure is:

`f:` the "stayopen" indicator, if non-zero, then the database will remain open over subsequent calls to the library.

The routine `endisobject` closes the database.

```
int     endisobject ()
```

Both of these routines return non-zero on success and zero otherwise.

There are two routines used to fetch a particular entry in the database. The routine `getisobjectbyname` maps object descriptors into the internal form.

```
struct isobject *getisobjectbyname (descriptor)
char   *descriptor;
```

The parameter to this procedure is:

`descriptor:` the descriptor of the object.

and returns the `isobject` structure describing that object. On failure, the manifest constant `NULL` is returned instead.

The routine `getisobjectbyoid` performs the inverse function.

```
struct isobject *getisobjectbyoid (oid)
OID     oid;
```