Part III

Databases

Chapter 5

The ISODE Services Database

The database *isoservices* in the ISODE `ETCDIR` directory (usually */usr/etc/*) contains a simple mapping between textual descriptions of services, service selectors, and local programs.

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

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

- the name of an entity and the provider on which the entity resides;
- the selector used to identify the entity to the provider, interpreted as a:

  number, if the selector starts with a hash-mark (‘`#`’). More precisely, this denotes the so-called GOSIP method for denoting selectors, which uses a two octet, network byte-order representation.

  ascii string, if the selector appears in double-quotes (‘`"`’). The usual escape mechanisms can be used to introduce non-printable characters.

  octet string, if all else fails. The standard “explosion” encoding is used, each octet in the string is represented by a two-digit hexadecimal quantity.

and,

- the program and argument vector to *execve*(2) when the service is requested.

Blanks and/or tab characters are used to separate items. All items after the first two are interpreted as an argument vector. 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.

5.1 Accessing the Database

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

```
struct isoservent {
    char   *is_entity;
    char   *is_provider;

#define ISSIZE  64
    int     is_selectlen;
    char    is_selector[ISSIZE];

    char  **is_vec;
    char  **is_tail;
};
```

The elements of this structure are:

`is_entity`: the name of the entity;

`is_provider`: the name of the provider on which the entity resides;

`is_selectoris_selectlen`: the selector used to identify the entity to the provider (the element `is_port` is an alias for this concept, used to denote the entity to the provider by means of a two-octet number specified in network-byte order);

`is_vec`: the *execve*(2) vector; and,

`is_tail`: the next free slot in `is_vec`.

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

```
struct isoservent *getisoservent ()
```

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

The routine `setisoservent` opens and rewinds the database.

```
int     setisoservent (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 `endisoservent` closes the database.

```
int     endisoservent ()
```

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 `getisoserventbyname` maps textual descriptions into the internal form.

```
struct isoservent *getisoserventbyname (entity, provider)
char   *entity,
       *provider;
```

The parameters to this procedure are:

`entity`: the entity providing the desired service; and,

`provider`: the provider supporting the named `entity`.

On a successful return, the `isoservent` structure describing that service is returned. On failure, the manifest constant `NULL` is returned instead.

The routine `getisoserventbyselector` performs the inverse function.

```
struct isoservent *getisoserventbyselector (provider,
                        selector, selectlen)
char   *provider,
       *selector;
int     selectlen;
```

The parameters to this procedure are:

`provider:` the provider supporting the desired entity; and,

`selector/selectlen:` the selector on the provider where the desired entity resides.

On a successful return, an `isoservent` structure describing the entity residing on the provider is returned.

The routine `getisoserventbyport` performs a similar function.

```
struct isoservent *getisoserventbyport (provider, port)
char   *provider;
unsigned short port;
```

The parameters to this procedure are:

`provider:` the provider supporting the desired entity; and,

`port:` the port on the provider (in network-byte order) where the desired entity resides.

On a successful return, an `isoservent` structure describing the entity residing on the provider is returned.

Chapter 6

The ISODE Tailoring File

The file *isotailor* in the ISODE `ETCDIR` directory (usually */usr/etc/*) contains a simple run-time configuration mechanism for programs loaded with the `-lisode` library.

The file itself is an ordinary ASCII text file containing information regarding the known tailoring options. Each line contains the option's name, a colon, the option's value, and a newline. The sharp character ('`#`') at the beginning of a line indicates a commentary line.

6.1 Tailor Variables

The options available, along with default values and a description of their meanings, are now described.

6.1.1 Local Environment Tailoring

`localname:` The name of the localhost. If not set, depending on the TCP/IP implementation you are running, the system will be queried for this value.

`binpath:` Where user programs are found.

`sbinpath:` Where system programs are bound.

`etcpath:` Where configuration files are found.

(continued on the next page)

6.1.2 Logging Tailoring

`logpath:` Where log files are found.

xyz`level:` The debugging level for the *xyz* module, defaulting to `none`.

Option	Module
`compatlevel`	*libicompat*(3n)
`addrlevel`	various
`tsaplevel`	*libtsap*(3n)
`ssaplevel`	*libssap*(3n)
`psaplevel`	*libpsap*(3)
`psap2level`	*libpsap2*(3n)
`acsaplevel`	*libacsap*(3n)
`rtsaplevel`	*librtsap*(3n)
`rosaplevel`	*librosap*(3n)

Options are seperated by whitespace. The debugging options available are:

`fatal:` fatal errors;

`exceptions:` exceptional events;

`notice:` informational notices;

`pdus:` *xyz*PDU printing;

`trace:` program tracing;

`debug:` full debugging (not fully defined at this point); and,

`all:` all of the above.

Logging of levels other than `fatal`, `exceptions`, or `notice` are subject to conditional compilation (the `-DDEBUG` option must be used during compilation).

xyz`file:` The file to be used for *xyz*PDU tracing. The file is written in append mode. If the filename supplied is '`-`' (a single dash), then

the diagnostic output is used instead.

Option	Default	Module
compatfile	%d.log	*libicompat*(3n)
addrfile	%d.log	various
tsapfile	%d.tpkt	*libtsap*(3n)
ssapfile	%d.spkt	*libssap*(3n)
psapfile	%d.pe	*libpsap*(3)
psap2file	%d.ppkt	*libpsap2*(3n)
acsapfile	%d.acpkt	*libacsap*(3n)
rtsapfile	%d.rtpkt	*librtsap*(3n)
rosapfile	%d.ropkt	*librosap*(3n)

The *getpid*(2) call is used to supply the value for %d.

6.1.3 Directory Services Tailoring

ns_enable: enables use of a "user-friendly nameservice" to perform name/address resolution. This takes the value either "on" or "off". If "on", then an OSI Directory-based service will be use. If the nameservice lookup fails, the stub-directory will be used as a fallback.

ns_address: the transport address of the nameservice. It is specified using the ISODE "string" format, e.g.,

```
Internet=wp.psi.net+17006
```

which indicates that the nameservice lives in the TCP/IP communications domain on TCP port 17006 at host wp.psi.net. The nameservice is accessed via the OSI CO-mode transport service, so other kinds of addresses (e.g., X.25 addresses can be used as well).

6.1.4 Transport Switch Tailoring

The use of these variables is more usefully described in Chapter 8.

ts_stacks: Specifies which configured TS-stacks are enabled. This is useful when multiple machines (with different interfaces) share the same executables. Options are seperated by whitespace: