`tcp`: RFC1006 over TCP/IP;

`x25`: TP0 over X.25;

`cons`: TP0 over CONS;

`bridge`: TP0–bridge;

`tp4`: TP4 over CLNP; and,

`all`: all of the above.

Using this method, the *isotailor* file is a normally symbolic link to */private/etc/isotailor*.

`ts_interim`: Defines new OSI communities. Each community is defined by a macro in the *i*somacros(5) file.

`ts_communities`: Specifies which OSI communities are attached (either directly or through a transport-service bridge). Options are seperated by whitespace:

`int-x25`: International X.25;

`janet`: UK JANET;

`internet`: the capital-I Internet;

`realns`: OSI Internet (ha, ha);

`localTCP`: the TCP loopback address; and,

`all`: all of the above OSI communities, along with those communities defined by the `ts_interim` variable;

For example, a site with an X.25 connection might be attached to the International X.25 network, but not the JANET. Thus `ts_stacks` would include "x25", and `ts_communities` would include "int-x25" but not "janet".

Note that the ordering of communities is important: network addresses will be tried in the order that their respective communities are listed with this variable.

`default_nsap_community`: the default community to be used for NSAP addresses.

`default_x25_community:` the default community to be used for X.25 (DTE) addresses.

`default_tcp_community:` the default community to be used for TCP (RFC1006) addresses.

### Transport-Service Bridge

There are two variables that can be specified. One is used on hosts making use of the TS-bridge, the other is used by hosts which run the TS-bridge:

`tsb_communities` A list of pairs of values. The first of each value should be a community name as defined in the `ts_communities` variable. The second value of the pair should be a presentation address using the ISODE "string" format. When a call is to be placed and the network corresponds to one of the communities given here, then a call through the TS-bridge given in the second variable will be made automatically.

`tsb_default_address` This variable contains a string encoded presentation address which the TS-bridge will listen on by default. This should normally consist of a set of network addresses with no selectors present.

Consider the case of a host with access to both the Internet and the International X.25 network. This host might have this entry in its *isotailor* file:

```
tsb default address: Internet=sheriff+17004"—Int-X25(80)=23426020017299+PID+03018000
```

This tells the TS-bridge to listen on two network endpoints. Hosts in the Internet community wishing to reach the International X.25 community would have this entry in their *isotailor* file:

```
tsb_communities: int-x25 Internet=sheriff+17004
```

Similarly, hosts in the International X.25 community wishing to reach the Internet community, would have the entry:

```
tsb_communities: internet Int-X25(80)=23426020017299+PID+03018000
```

### 6.1.5 Interface Specific Tailoring

Some network implementations used by ISODE require ISODE to be tailored for their correct use.

#### General X.25 Tailoring

The following tailoring variables are generally applicable to X.25 networks.

`x25_local_dte` It is normally necessary for ISODE to know it's local DTE address. This variable is used to set this. The default is empty, i.e., do not set a calling address in call requests.

`x25_local_pid` It is normally necessary for ISODE to know the X.25 protocol ID to listen on This is specified in hex-notation, e.g., `03010100`.

`x25_intl_zero` Some Public Data Networks require that X.121 addresses be modified before being conveyed. If this variable has the value `on` then any addresses with a non-local DNIC will have a leading zero appended.

`x25_strip_dnic` If this variable has the value `on` then any address with a local DNIC will have this removed.

`x25_dnic_prefix` If you use either or both of the preceding two mechanisms then you must use this variable to inform ISODE of the local DNIC for your host.

`x25level:` Defines the level of logging to be used for X.25 statistics logging. (At present, only `notice` messages are generated.)

`x25file:` Defines the filename to be used for X.25 statistics logging.

#### SunLink X.25

The following variables are currently only supported by the SunLink X.25 interface. They control the X.25 Facilities that are requested or accepted.

`reverse_charge` If `0` then don't request/allow (initiator/responder) reverse charging. If `1` then request/allow reverse charging.

`recvpktsize/sendpktsize` Size of level 3 packets. Valid values are `0` (default size), `16`, `32`, `64`, `128`, `256`, `512`, `1024` (octets in decimal).

`recvwndsize/sendwndsize` Size of level 3 window. Valid values are `0` (default window size), `1`–`7` or `1`–`127` (decimal).

`recvthruput/sendthruput` Send/receive throughput.

| | | | |
|---|---|---|---|
| `0` | default throughput | | |
| `3` | 75 | `8` | 2400 |
| `4` | 150 | `9` | 4800 |
| `5` | 300 | `10` | 9600 |
| `6` | 600 | `11` | 19200 |
| `7` | 1200 | `12` | 48000 |

Values in bps decimal.

`cug_req` If `0` then don't use closed user group, if `1` then use closed user group specified by `cug_index`.

`cug_index` Closed user group in decimal (`00`–`99`).

`fast_select_type` `0` = don't use/allow fast select. `1` = calling side — only accept clear in response to fast select, called side — send clear in response to fast select. `2` = clear or accept is valid response to fast select.

`rpoa_req` If `0` then don't request RPOA (Recognised Private Operating Agency) transit. If `1` then request RPOA transit.

`rpoa` If `rpoa_req` is `1` then this is RPOA transit group in decimal (`0000`–`9999`).

See the SunLink X.25 Programmer's Manual for further explanations of these facilities.

### Camtec CCL

There is one tailoring variable for the Camtec X.25 when used with the socket abstraction.[1]

`x25_outgoing_port` selects the physical port on the Camtec card for outgoing calls. It may take the value `A`, `B` or `#`. `A` and `B` are the X.121 WAN ports and `#` is the IEEE 802.3 (Ethernet) port. Incoming calls will be accepted on any port.

### Bridge X.25

There are several tailorable variables that can specified for the bridge connection. These are:

`x25_bridge_host` selects the host that runs the tp0bridge being used. This should be a TCP accessible host.

`x25_bridge_port` selects the TCP port that the tp0bridge will be listening on. The default for this is port 146 (an internet assigned number), which should be defined in `/etc/services`.

`x25_bridge_addr` the X.121 address of the remote host.

`x25_bridge_listen` the X.121 address that this host will be listening on for incoming calls via the bridge.

`x25_bridge_pid` the protocol id used for listening along with the above address. This is a set of eight hex digits.

`x25_bridge_discrim` selects the network layer to use. When attempting to place a call with the bridge code configured as well as real X.25, the string selects the interface to use. If the string is empty, the bridge will always be used. If it is set to "–" the bridge will not be used. If the string is anything else, it is compared against the initial portion of the called X.121 address. If there is a match then the bridge is used, otherwise the real interface is called.

[1] The old device level interface is no longer supported.

## 6.2 Accessing the Tailoring File

The tailoring file is read usually when a program attempts or accepts its first connection. The `-lisode` library does this by calling the routine `isodetailor`:

```
void    isodetailor (myname, wantuser)
char   *myname;
int     wantuser;
```

The parameters to this procedure are:

`myname:` the name that the program was invoked with (used by the logging package described in Chapter 7); and,

`wantuser:` if non-zero, then a user-specific tailoring file, with the name `~/.myname_tailor`, should be consulted.

Note that in order to ensure consistent logging it is **critical** that the call to `isodetailor` be the first call made to *any* of the ISODE routines.

To override the default location of the tailoring file, use the routine `isodesetailor`:

```
char   *isodesetailor (file)
char   *file;
```

The parameter to this procedure is:

`file:` the filename to be used instead of the default. Future versions of this routine might act differently.

The filename is interpreted relative to the `-lisode` system area. To override this, specify a anchored pathname (e.g., on UNIX, one which starts with `/` or `./`). The routine returns the name of the default tailoring file.

To set a tailoring variable from some other configuration file, the routines `isodesetvar` and `isodexport` are used:

```
int     isodesetvar (name, value, dynamic)
char   *name,
       *value;
int     dynamic;
```

When this routine is invoked, it acts as though

```
name: value
```

was found in the tailoring file. The `dynamic` parameter, if non-zero, indicates that `value` may be freed if a subsequent call to `isodesetvar` is made which overrides the previous value.

The `isodexport` routine is called after one or more calls to `isodesetvar`, it performs any post-processing necessary to resynchronize the tailoring facilities.

```
void    isodexport ()
```

Thus, to read a private tailoring file, `isodesetvar` should be called for each tailoring line. Then, `isodexport` should be called once to resynchronize things.

Finally, it may be necessary to access files in the `-lisode` system area. The routine `isodefile` takes an filename and returns an anchored pathname.

```
char   *isodefile (file, ispgm)
char   *file;
int ispgm;
```

The parameters to this procedure are:

`file`: the filename to be expanded; and,

`ispgm`: non-zero if the target file is an executable (otherwise it is a database of some kind).

This routine is actually a macro which invokes the routine `_isodefile`:

```
char   *_isodefile (path, file)
char   *path,
       *file;
```

The parameters to this procedure are:

`path`: the directory where the filename should be expanded; and,

`file`: the filename to be expanded.

# Chapter 7

# The ISODE Logging Facility

Although not a database mechanisms, per se, the ISODE logging facility is used to manipulate general logs: used by both the ISODE and programs which use the ISODE.

### 7.0.1 Data-Structures

There is one primary data-structure, the LLog:

```
typedef struct  ll_struct {
    char   *ll_file;

    char   *ll_hdr;
    char   *ll_dhdr;

    int     ll_events;
#define LLOG_NONE 0
#define LLOG_FATAL 0x01
#define LLOG_EXCEPTIONS 0x02
#define LLOG_NOTICE 0x04
#define LLOG_PDUS 0x08
#define LLOG_TRACE 0x10
#define LLOG_DEBUG 0x20
#define LLOG_ALL 0xff
#define LLOG_MASK \
    "\020\01FATAL\02EXCEPTIONS\03NOTICE\04PDUS\05TRACE\06DEBUG"
```

```
    int     ll_syslog;

    int     ll_msize;

    int     ll_stat;
#define LLOGNIL 0x00
#define LLOGCLS 0x01
#define LLOGCRT 0x02
#define LLOGZER 0x04
#define LLOGERR 0x08
#define LLOGTTY 0x10
#define LLOGHDR 0x20
#define LLOGDHR 0x40

    int     ll_fd;
} LLog;
```

The elements of this structure are:

`ll_file`: the name of the file to use for the log, unless an absolute pathname (e.g., */usr/tmp/logfile*) or an anchored pathname (e.g., *./logfile*), the name is interpreted relative to the the `logpath` directory in the ISODE tailoring file (see Chapter 6);

`ll_hdr`: the logging header which is usually set by one of the utility routines described below;

`ll_hdr/ll_dhdr`: the so-called dynamic header;

`ll_events/ll_syslog`: a bitmask describing the logging events which are interesting to this log, any combination of:

| Value | Meaning |
|---|---|
| LLOG_FATAL | fatal errors |
| LLOG_EXCEPTIONS | exceptional events |
| LLOG_NOTICE | informational notices |
| LLOG_PDUS | PDU printing |
| LLOG_TRACE | program tracing |
| LLOG_DEBUG | full debugging |

In addition, the values `LLOG_NONE` by itself refers to no events and `LLOG_ALL` refers to all events being of interest. For those systems with a *syslog*(3) routine, the `ll_syslog` element indicates if the event should be given to *syslog*(8) as well;

`ll_msize`: the maximum size of the log, in units of Kbytes (a non-positive number indicates no limit);

`ll_stat`: assorted switches, any combination of:

| Value | Meaning |
|---|---|
| `LOGCLS` | keep log closed, except when writing |
| `LOGCRT` | create log if necessary |
| `LOGZER` | truncate log when limits reached |
| `LOGERR` | log closed due to (soft) error |
| `LOGTTY` | also log to stderr |
| `LOGHDR` | static header allocated |
| `LOGDHR` | dynamic header allcoated |

`ll_fd`: the file-descriptor corresponding to the log.

## 7.1 Accessing the Log

Typically, logs are not opened or closed directly — when an entry is made to a log, the log is opened (if necessary), the entry is written, and (usually) the log is then closed.

To open a log associated with a `LLog` structure, the routine `ll_open` is used:

```
int     ll_open (lp)
LLog   *lp;
```

The parameter to this routine is:

`lp`: a pointer to a `LLog` structure.

The `ll_open` routine will open the log, creating the corresponding file (if necessary). Logs are created mode `0666`. If the name of the file to use for