and the abstract syntax (a pointer to an object identifier, see Section 5.4.6 of *Volume One*). The provider selects the abstract transfer notation (in the current implementation, this is always ASN.1[ISO87a]). When the a P-CONNECT.INDICATION event is given to the responder, in addition indicating the context identifier and abstract syntax information, the provider also indicates if it is willing to support this presentation context. If so, the responder decides if it will accept or reject the context. This information is then propagated back to the initiator with the P-CONNECT.CONFIRMATION indication.

2.3.2 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 SSAP entity. In the case of the presentation service, the *tsapd* program contains the bootstrap for the presentation provider. The daemon will again consult the ISO services database to determine which program on the system implements the desired PSAP 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 PSAP state contained in the magic arguments. This is done by calling the routine `PInit`, which on a successful return, is equivalent to a P-CONNECT.INDICATION from the presentation service provider.

```
int     PInit (vecp, vec, ps, pi)
int     vecp;
char  **vec;
struct PSAPstart *ps;
struct PSAPindication *pi;
```

The parameters to this procedure are:

`vecp`: the length of the argument vector;

`vec`: the argument vector;

`ps`: a pointer to a `PSAPstart` structure, which is updated only if the call succeeds; and,

`pi`: a pointer to a `PSAPindication` structure, which is updated only if the call fails.

If `PInit` is successful, it returns information in the `ps` parameter, which is a pointer to a `PSAPstart` structure.

```
struct PSAPstart {
    int     ps_sd;

    struct PSAPaddr ps_calling;
    struct PSAPaddr ps_called;

    struct PSAPctxlist ps_ctxlist;

    OID     ps_defctx;
    int     ps_defctxresult;

    int     ps_prequirements;

    int     ps_srequirements;
    int     ps_settings;
    long    ps_isn;

    struct SSAPref  ps_connect;

    int     ps_ssdusize;

    struct QOStype ps_qos;

#define NPDATA          10
    int     ps_ninfo;
    PE      ps_info[NPDATA];
};
```

The elements of this structure are:

`ps_sd`: the presentation-descriptor to be used to reference this connection;

`ps_calling`: the address of the peer initiating the connection;

`ps_called`: the address of the peer being asked to respond;

`ps_ctxlist`: the proposed list of presentation contexts;

`ps_defctx/ps_defctxresult`: the default context for the presentation connection (and the presentation provider's response);

`ps_prequirements`: the proposed presentation requirements;

`ps_srequirements`: the proposed session requirements;

`ps_settings`: the initial token settings;

`ps_isn`: the initial serial-number;

`ps_connect`: the connection identifier (a.k.a. SSAP reference) used by the initiator;

`ps_ssdusize`: the largest atomic SSDU size that can be used on the connection (see the note on page 45);

`ps_qos`: the quality of service on the connection (see Section 4.6.2); and,

`ps_info/ps_ninfo`: an array of initial data (and the number of elements in that array).

Note that the `ps_info` element is allocated via *malloc*(3) and should be released using the `PSFREE` macro when no longer referenced. The `PSFREE` macro behaves as if it was defined as:

```
void    PSFREE (ps)
struct PSAPstart *ps;
```

The macro frees only the data allocated by `PInit`, and not the `PSAPstart` structure itself. Further, `PSFREE` should be called only if the call to the `PInit` routine returned `OK`.

The `ps_connect` element is a `SSAPref` structure, which is passed transparently by the presentation service. Consult the description on page 52. There are two routines of interest in the *libpsap2*(3n) that deal with these structures: The `addr2ref` routine takes a string (presumably a hostname), determines the current UT time, and returns a pointer to a `SSAPref` structure appropriately initialized to denote this information.

```
struct SSAPref *addr2ref (addr)
char   *addr;
```

This routine might fail if it is unable to allocate a small amount of memory. In this event, it returns the manifest constant `NULL`. The routine `sprintref` can be used to return a null-terminated string describing the SSAP reference.

```
char   *sprintref (sr)
struct SSAPref *sr;
```

The `ps_ctxlist` element is a `PSAPctxlist` structure, which describes the presentation contexts the initiator wishes to use.

```
struct PSAPctxlist {
    int     pc_nctx;

#define NPCTX   5

    struct PSAPcontext {
        int     pc_id;
        OID     pc_asn;
        OID     pc_atn;
        int     pc_result;
    }       pc_ctx[NPCTX];
};
#define NULLPC  ((struct PSAPctxlist *) 0)
```

The elements of this structure are:

`pc_ctx/pc_nctx:` the presentation contexts described (and the number of contexts which may not exceed `NPCTX` elements). For each presentation context:

`pc_id:` the identifier (or handle) for the context;

`pc_asn:` the abstract syntax notation to be used on the context;

`pc_atn:` the transfer syntax notation to be used on the context (this field is usually, `NULLOID`, only the initiator, when it wishes to inform the presentation service of the transfer syntax to use, is permitted to specify this); and,

`pc_result`: the presentation provider's response (codes are listed in Table 2.1).

If the call to the `PInit` routine is not successful, then a P-P-ABORT.INDICATION event is simulated, and the relevant information is returned in a `PSAPindication` structure.

```
struct PSAPindication {
    int     pi_type;
#define PI_DATA         0x00
#define PI_TOKEN        0x01
#define PI_SYNC         0x02
#define PI_ACTIVITY     0x03
#define PI_REPORT       0x04
#define PI_FINISH       0x05
#define PI_ABORT        0x06

    union {
        struct PSAPdata pi_un_data;
        struct PSAPtoken pi_un_token;
        struct PSAPsync pi_un_sync;
        struct PSAPactivity pi_un_activity;
        struct PSAPreport pi_un_report;
        struct PSAPfinish pi_un_finish;
        struct PSAPabort pi_un_abort;
    }   pi_un;
#define pi_data         pi_un.pi_un_data
#define pi_token        pi_un.pi_un_token
#define pi_sync         pi_un.pi_un_sync
#define pi_activity     pi_un.pi_un_activity
#define pi_report       pi_un.pi_un_report
#define pi_finish       pi_un.pi_un_finish
#define pi_abort        pi_un.pi_un_abort
};
```

As shown, this structure is really a discriminated union (a structure with a tag element followed by a union). Hence, on a failure return, one first coerces a pointer to the `PSAPabort` structure contained therein, and then consults the elements of that structure.

```
struct PSAPabort {
    int     pa_peer;

    int     pa_reason;
    int     pa_ppdu;


    int     pa_ninfo;
    char    pa_info;

#define PA_SIZE        512
    int     pa_cc;
    char    pa_data[PA_SIZE];
};
```

The elements of a `PSAPabort` structure are:

`pa_peer`: if set, indicates that a user-initiated abort occurred (a P-U-ABORT.INDICATION event); if not set, indicates that a provider-initiated abort occurred (a P-P-ABORT.INDICATION event);

`pa_reason`: the reason for the provider-initiated abort (codes are listed in Table 2.1);

`pa_ppdu`: the type of presentation protocol data unit which triggered the provider-initiated abort (codes are listed in Table 2.2);

`pa_data/pa_cc`: a provider-generated diagnostic string (and the length of that string); and,

`pa_info/pa_ninfo`: an array of user data, and the number of elements in that array (if `pa_peer` is set).

Note that the `pa_info` element is allocated via *malloc*(3) and should be released using the `PAFREE` macro when no longer referenced. The `PAFREE` macro behaves as if it was defined as:

```
void    PAFREE (pa)
struct PSAPabort *pa;
```

Provider-initiated Abort (fatal)	
PC_ADDRESS	Called presentation address unknown
PC_AVAILABLE	No PSAP available from those identified presentation address
PC_CONGEST	Local limit exceeded
PC_VERSION	Protocol version not supported
PC_UNSPECIFIED	Unspecified
PC_UNRECOGNIZED	Unrecognized PPDU
PC_UNEXPECTED	Unexpected PPDU
PC_SSPARAM	Unexpected session service parameter
PC_PPPARAM	Unrecognized PPDU parameter
PC_INVALID	Invalid PPDU parameter
PC_ABSTRACT	Abstract syntax not supported
PC_TRANSFER	Proposed transfer syntaxes not supported
PC_REFUSED	Connect request refused on this network connection
PC_SESSION	Session disconnect
PC_PROTOCOL	Protocol error
PC_ABORTED	Peer aborted connection
User-initiated Abort (fatal)	
PC_REJECTED	Rejected by peer
Interface Errors (non-fatal)	
PC_PARAMETER	Invalid parameter
PC_OPERATION	Invalid operation
PC_TIMER	Timer expired
PC_WAITING	Indications waiting

Table 2.1: PSAP Failure Codes

PPDU	Associated Operation
PPDU_NONE	none
PPDU_CP	connection
PPDU_CPA	connection accept
PPDU_CPR	connection reject
PPDU_ARU	user abort
PPDU_ARP	provider abort
PPDU_AC	alter context
PPDU_ACA	alter context ack
PPDU_TD	data
PPDU_TTD	typed-data
PPDU_TE	expedited data
PPDU_TC	capability data
PPDU_TCC	capability data ack
PPDU_RS	resynchronize
PPDU_RSA	resynchronize ack

Table 2.2: PSAP PPDU Codes

The macro frees only the data allocated in the PSAPabort structure and not the structure itself.

After examining the information returned by PInit on a successful call (and possibly after examining the argument vector), the responder should either accept or reject the connection. For either response, the responder should use the PConnResponse routine (which corresponds to the P-CONNECT.RESPONSE action).

```
int     PConnResponse (sd, status, responding, ctxlist,
                defctxresult, prequirements, srequirements,
                isn, settings, ref, data, ndata, pi)
int     sd;
struct PSAPaddr *responding;
int     status,
        prequirements,
        srequirements,
        settings,
        ndata;
long isn;
struct PSAPctxlist *ctxlist;
int     defctxresult;
struct SSAPref *ref;
PE     *data;
struct PSAPindication *pi;
```

The parameters to this procedure are:

sd: the presentation-descriptor;

result: the acceptance indicator (either PC_ACCEPT if the connection is to be accepted, or PC_REJECTED otherwise);

responding: the PSAP address of the responder (defaulting to the called address, if not present);

ctxlist: the responder's decision as to each of the presentation contexts suggested (for each proposed context, if the pc_result element supplied by the provider is PC_ACCEPT, which indicates that the provider is willing to support it, the user may supply either PC_ACCEPT or the value PC_REJECTED);

`defctxresult`: the response to the default context (if the presentation provider responded with `PC_ACCEPT`, the user may supply either `PC_ACCEPT` or `PC_REJECTED`);

`prequirements`: the responder's presentation requirements;

`srequirements`: the responder's session requirements;

`isn`: the initial serial-number;

`settings`: the initial token settings;

`ref`: the connection identifier used by the responder (consult page 52 for a description of the `SSAPref` structure);

`data/ndata`: an array of initial data (and the number of elements in that array, consult the warning on page 35); and,

`pi`: a pointer to a `PSAPindication` structure, which is updated only if the call fails.

If the call to `PConnResponse` is successful, and if the `result` parameter is set to `PC_ACCEPT`, then connection establishment has completed and the users of the presentation service now operate as symmetric peers. If the call is successful, but the `result` parameter is `PC_REJECTED` instead, then the connection has been rejected and the responder may exit. Otherwise, if the call fails and the reason is not an interface error (see Table 2.1 on page 25), then the connection is closed.

Note that when the responder rejects the connection, it need only supply meaningful values for the `sd`, `status`, `defctxresult`, and `pi` parameters.

2.3.3 Client Initialization

A program wishing to connect to another user of presentation services calls the `PConnRequest` routine, which corresponds to the P-CONNECT.REQUEST action.

```
int     PConnRequest (calling, called, ctxlist, defctxname,
                      prequirements, srequirements, isn, settings,
                      ref, data, ndata, qos, pc, pi)
```