`rts_sd`: the association-descriptor to be used to reference this association;

`rts_mode`: the dialogue mode (either monologue or two-way alternate),

`rts_turn`: the owner of the turn initially;

`rts_data`: any initial data (this is a pointer to a `PElement` structure, which is fully explained in Chapter 5); and,

`rts_start`: a `AcSAPstart` structure (consult page 26).

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

```
void    RTSFREE (rts)
struct RtSAPstart *rts;
```

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

If the call to `RtInit` is not successful, then a RT-P-ABORT.INDICATION event is simulated, and the relevant information is returned in an encoded `RtSAPindication` structure (discussed in Section 4.2 on page 95).

After examining the information returned by `RtInit` on a successful call (and possibly after examining the argument vector), the responder should either accept or reject the association. For either response, the responder should use the `RtOpenResponse` routine (which corresponds to the RT-OPEN.RESPONSE action).

```
int     RtOpenResponse (sd, status, context, respondtitle,
        respondaddr, ctxlist, defctxresult, data, rti)
int     sd,
        status;
OID     context;
AEI     respondtitle;
struct PSAPaddr *respondaddr;
struct PSAPctxlist *ctxlist;
int     defctxresult;
PE      data;
struct RtSAPindication *rti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`status:` the acceptance indicator (either `ACS_ACCEPT` if the association is to be accepted, or one of the user-initiated rejection codes listed in Table 2.1 on page 29);

`data:` any initial data (regardless of whether the association is to be accepted); and,

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

The remaining parameters are for the association control service, consult the description of the `AcAssocResponse` routine on page 30.

If the call to `RtOpenResponse` is successful, and the `status` parameter was set to `ACS_ACCEPT`, then association establishment has now been completed. If the call was successful, but the `status` parameter was not `ACS_ACCEPT`, then the association has been rejected and the responder may exit. Otherwise, if the call failed and the reason is "fatal", then the association is lost.

Client Initialization

A program wishing to associate itself with another user of reliable transfer services calls the `RtOpenRequest` routine, which corresponds to the RT-OPEN.REQUEST action.

```
int     RtOpenRequest (mode, turn, context, callingtitle,
        calledtitle, callingaddr, calledaddr, ctxlist,
        defctxname, data, qos, rtc, rti)
int     mode,
        turn;
AEI     callingtitle,
        calledtitle;
OID     context;
struct PSAPaddr *callingaddr,
                *calledaddr;
int     single;
```

```
struct PSAPctxlist *ctxlist;
OID     defctxname;
PE      data;
struct QOStype *qos;
struct RtSAPconnect *rtc;
struct RtSAPindication *rti;
```

The parameters to this procedure are:

`mode`: the dialogue mode;

`turn`: who gets the initial turn;

`data`: any initial data;

`qos`: the quality of service on the connection (see Section 4.6.2 in *Volume Two*);

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

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

The remaining parameters are for the association control service, consult the description of the `AcAssocRequest` routine on page 32. If the call to `RtOpenRequest` is successful (this corresponds to a RT-OPEN.CONFIRMATION event), it returns information in the `rtc` parameter, which is a pointer to a `RtSAPconnect` structure.

```
struct RtSAPconnect {
    int     rtc_sd;

    int     rtc_result;

    PE      rtc_data;

    struct AcSAPconnect rtc_connect;
};
```

The elements of this structure are:

`rtc_sd`: the association-descriptor to be used to reference this association;

`rtc_result`: the association response;

`rtc_data`: any initial data (regardless of whether the association was accepted); and,

`rtc_connect`: a `AcSAPconnect` structure (consult page 33).

If the call to `RtOpenRequest` is successful, and the `rtc_result` element is set to `RTS_ACCEPT`, then association establishment has completed. If the call is successful, but the `rtc_result` element is not `RTS_ACCEPT`, then the association attempt has been rejected; consult Table 4.1 to determine the reason for the reject. Otherwise, if the call fails then the association is not established and the `RtSAPabort` structure of the `RtSAPindication` discriminated union has been updated.

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

```
void    RTCFREE (rtc)
struct RtSAPconnect *rtc;
```

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

4.1.2 Association Release

The `RtCloseRequest` routine is used to request the release of an association, and corresponds to a RT-CLOSE.REQUEST action.

```
int     RtCloseRequest (sd, reason, data, acr, rti)
int     sd,
        reason;
PE      data;
struct AcSAPrelease *acr;
struct RtSAPindication *rti;
```

The parameters to this procedure:

`sd:` the association-descriptor;

`reason:` the reason why the association should be released, one of:

Value	Reason
ACF_NORMAL	normal
ACF_URGENT	urgent
ACF_UNDEFINED	undefined

`data:` any final data;

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

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

If the call to `RtCloseRequest` is successful, then this corresponds to a RT-CLOSE.CONFIRMATION event, and it returns information in the `acr` parameter, which is a pointer to a `AcSAPrelease` structure (consult page 37).

If the call to `RtCloseRequest` is successful, then the association has been released. Otherwise the `RtSAPabort` element of the `RtSAPindication` parameter `rti` contains the reason for failure.

Upon receiving a RT-CLOSE.INDICATION event, the user is required to generate a RT-CLOSE.RESPONSE action using the `RtCloseResponse` routine.

```
int     RtCloseResponse (sd, reason, data, rti)
int     sd,
        reason;
PE      data;
struct RtSAPindication *rti;
```

The parameters to this procedure:

`sd:` the association-descriptor;

`reason:` the reason for the indicator, one of:

Value	Reason
ACR_NORMAL	normal
ACR_NOTFINISHED	not finished
ACR_UNDEFINED	undefined

`data:` any final data; and,

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

If the call to `RtCloseResponse` is successful, then the association has been released. If the call was successful, but the `reason` parameter was not `ACR_NORMAL`, then the association remains established.

4.1.3 Association Abort

To unilaterally abort an association, the routine `RtUAbortRequest` routine is used which corresponds to the RT-U-ABORT.REQUEST action.

```
int     RtUAbortRequest (sd, data, rti)
int     sd;
PE      data;
struct RtSAPindication *rti;
```

The parameters to this procedure:

`sd:` the association-descriptor;

`data:` any abort data; and,

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

If the call to `RtUAbortRequest` is successful, then the association is immediately relesaed, and any data queued for the association may be lost.

4.2 Reliable Transfer

> **NOTE:** Users should also consult Sectin 4.2.6 on page 104 for optional routines to make RTS behave more sanely.

Once the association has been established, an association-descriptor is used to reference the association. This is usually the first parameter given to any of the remaining routines in the *librtsap*(3n) library. Further, the last parameter is usually a pointer to a `RtSAPindication` structure. If a call to one of these routines fails, then the structure is updated.

```
struct RtSAPindication {
    int     rti_type;
#define RTI_TURN        0x00
#define RTI_TRANSFER    0x01
#define RTI_ABORT       0x02
#define RTI_CLOSE       0x03 (X.410 CLOSE)
#define RTI_FINISH      0x04 (RT-CLOSE)

    union {
        struct RrSAPturn rti_un_turn;
        struct RtSAPtransfer rti_un_transfer;
        struct RtSAPabort rti_un_abort;
        struct RtSAPclose rti_un_close;
        struct AcSAPfinish rti_un_finish
    }   rti_un;
#define rti_turn        rti_un.rti_un_turn
#define rti_transfer    rti_un.rti_un_transfer
#define rti_abort       rti_un.rti_un_abort
#define rti_close       rti_un.rti_un_close
#define rti_finish      rti_un.rti_un_finish
};
```

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 `RtSAPabort` structure contained therein, and then consults the elements of that structure.

```
struct RtSAPabort {
    int     rta_peer;

    int     rta_reason;

    PE      rta_udata;

#define RTA_SIZE        512
    int     rta_cc;
    char    rta_data[RTA_SIZE];
};
```

The elements of a `RtSAPabort` structure are:

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

`rta_reason`: the reason for the abort (codes are listed in Table 4.1);

`rta_udata`: optionally, data associated with the user-initiated abort; and,

`rta_data`/`rta_cc`: a diagnostic string from the provider.

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

```
void    RTAFREE (rta)
struct RtSAPabort *rta;
```

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

On a failure return, if the `rta_reason` element of the `RtSAPabort` structure is associated with a fatal error, the the association is released. The `RTS_FATAL` macro can be used to determine this.

```
int     RTS_FATAL (r)
int     r;
```

Provider-Initiated Aborts (fatal)	
RTS_ADDRESS	Address unknown
RTS_REFUSED	Connect request refused on this network connection
RTS_SESSION	Session disconnect
RTS_PROTOCOL	Protocol error
RTS_CONGEST	Congestion at RtSAP
RTS_REMOTE	Remote system problem
RTS_PRESENTATION	Presentation disconnect
RTS_ACS	ACS disconnect
RTS_ABORTED	Peer aborted association
User-Initiated Rejects (fatal)	
RTS_BUSY	Busy
RTS_RECOVER	Cannot recover
RTS_VALIDATE	Validation failure
RTS_MODE	Unacceptable dialogue mode
RTS_REJECT	Rejected by responder
Interface Errors (non-fatal)	
RTS_PARAMETER	Invalid parameter
RTS_OPERATION	Invalid operation
RTS_TIMER	Timer expired
RTS_WAITING	Indications waiting
RTS_TRANSFER	Transfer failure

Table 4.1: RtSAP Failure Codes

For protocol purists, the `RTS_OFFICIAL` macro can be used to determine if the error is an "official" error as defined by the specification, or an "unofficial" error used by the implementation.

```
int     RTS_OFFICIAL (r)
int     r;
```

4.2.1 Sending Data

When the user has the turn, it can use the `RtTransferRequest` routine (this corresponds to the RT-TRANSFER.REQUEST action) to request the reliable transfer of a data structure.

```
int     RtTransferRequest (sd, data, secs, rti)
int     sd;
PE      data;
int     secs;
struct RtSAPindication *rti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`data:` the data to be transferred;

`secs:` the amount of time, in seconds, permitted to transfer the data (use the manifest constant `NOTOK` if time is unimportant); and,

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

If `RtTransferRequest` succeeds, then the data has been reliably transferred to the other user. Otherwise the `RtSAPabort` structure contained in the `RtSAPindication` parameter `rti` contains the reason for failure.

4.2.2 Receiving Data

The `RtWaitRequest` routine is used to wait for an event (usually incoming data) to occur.