```
int     RtBInit (vecp, vec, rts, rti)
int     vecp;
char  **vec;
struct RtSAPstart *rts;
struct RtSAPindication *rti;
```

The parameters to this procedure are:

`vecp`: the length of the argument vector;

`vec`: the argument vector;

`rts`: a pointer to a `RtSAPstart` 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 `RtBInit` is successful, it returns information in the `rts` parameter, which is a pointer to a `RtSAPstart` structure.

```
struct RtSAPstart {
    int     rts_sd;

    struct RtSAPaddr rts_initiator;

    int     rts_mode;
#define RTS_MONOLOGUE   0
#define RTS_TWA         1

    int     rts_turn;
#define RTS_INITIATOR   0
#define RTS_RESPONDER   1

    u_short rts_port;

    PE      rts_data;
};
```

The elements of this structure are:

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

`rts_initiator`: the address of the initiator;

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

`rts_turn`: the owner of the turn initially;

`rts_port`: the application number the initiator wishes to use to govern the association; and

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

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 `RtBInit`, and not the `RtSAPstart` structure itself. Further, `RTSFREE` should be called only if the call to the `RtBInit` routine returned `OK`.

If the call to `RtBInit` is not successful, then a RT-P-ABORT.INDICATION event is simulated, and the relevant information is returned in an encoded `RtSAPindication` structure.

After examining the information returned by `RtBInit` 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 `RtBeginResponse` routine (which corresponds to the X.410 OPEN.RESPONSE action).

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

The parameters to this procedure are:

`sd`: the association-descriptor;

`status`: the acceptance indicator (either `RTS_ACCEPT` if the association is to be accepted, or one of the "fatal" user-initiated rejection codes, other tha `RTS_REJECT`, listed in Table 4.1 on page 97);

`data`: any initial data if the association is to be accepted; and,

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

If the call to `RtBeginResponse` is successful, and the `status` parameter is set to `RTS_ACCEPT`, then association establishment has now been completed. If the call is successful, but the `status` parameter is not `RTS_ACCEPT`, then the association has been rejected and the responder may exit. Otherwise, if the call fails and the reason is "fatal", then the association is closed.

Client Initialization

A program wishing to associate itself with another user of reliable transfer services calls the `RtBeginRequest` routine, which corresponds to the X.410 OPEN.REQUEST action.

```
int     RtBeginRequest (called, mode, turn, data, rtc, rti)
struct  RtSAPaddr *called;
int     mode,
        turn;
PE      data;
struct RtSAPconnect *rtc;
struct RtSAPindication *rti;
```

The parameters to this procedure are:

`called`: the RtSAP address of the responder;

`mode`: the dialogue mode;

`turn`: who gets the initial turn;

`data`: any initial data;

`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.

If the call to `RtBeginRequest` is successful (this corresponds to a X.410 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;
};
```

The elements of this structure are:

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

`rtc_result`: the association response; and,

`rtc_data`: any initial data if the association was accepted.

If the call to `RtBeginRequest` 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`, the 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 `RtBeginRequest`, and not the `RtSAPconnect` structure itself. Further, `RTCFREE` should be called only if the call to the `RtBeginRequest` routine returned `OK`.

A.2.3 Association Release

The `RtEndRequest` routine is used to request the release of an association, and corresponds to a X.410 CLOSE.REQUEST action. This action may be taken by only the initiator of an association, and, if the dialogue mode is two-way alternate, if the initiator has the turn as well.

```
int     RtEndRequest (sd, rti)
int     sd;
struct RtSAPindication *rti;
```

The parameters to this procedure are:

`sd:` the association-descriptor; and,

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

If the call to `RtEndRequest` is successful, then the association has been released. Otherwise if the call fails and the error is not fatal, then the association remains established and the `RtSAPabort` structure contained in the `RtSAPindication` parameter `rti` contains the reason for the failure.

Upon receiving a X.410 CLOSE.INDICATION event, the user is required to generate a X.410 CLOSE.RESPONSE action using the `RtEndResponse` routine.

```
int     RtEndResponse (sd, rti)
int     sd;
struct RtSAPindication *rti;
```

The parameters to this procedure are:

`sd:` the association-descriptor; and,

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

If the call to `RtEndResponse` is successful, then the association has been released.

A.2.4 An Example

Let's consider how one might construct a generic server that uses reliable transfer services to establish an association, but then uses remote operation services to communicate with its peer. This entity will use a synchronous interface.

There are two parts to the program: initialization and the request/reply loop. In our example, we assume that the routine `error` results in the process being terminated after printing a diagnostic.

In Figure A.5, the initialization steps for the generic responder, including the outer *C* wrapper, is shown. First, the RtSAP state is re-captured by calling `RtBInit`. If this succeeds, then the association is authenticated and any command line arguments are parsed. Assuming that the responder is satisfied with the proposed association, it calls `RtBeginResponse` to accept the association. The `RoSetService` routine is called to set the underlying service to be used for remote operations. Finally, the main request/reply loop is realized. The responder calls `RoWaitRequest` to get the next event, and then calls `ros_indication` to decode that event.

Figure 3.2 on page 77 contains the definition of the `ros_indication` routine and associated routines. Since the reliable transfer service was used to establish the association, a different closing handler must be used. This is shown in Figure A.6.

```
#include <stdio.h>
#include <isode/rtsap.h>
#include <isode/isoservent.h>


main (argc, argv, envp)
int       argc;
char  **argv,
     **envp;
{                                                                    10
    int       result,
              sd;
    struct RoSAPindication  rois;
    register struct RoSAPindication *roi = &rois;
    register struct RoSAPpreject   *rop = &roi -> roi_preject;
    struct RtSAPstart   rtss;
    register struct RtSAPstart *rts = &rtss;
    struct RtSAPindication  rtis;
    register struct RtSAPindication *rti = &rtis;
    register struct RtSAPabort   *rta = &rti -> rti_abort;           20

    if (RtInit (argc, argv, rts, rti) == NOTOK)
        error ("initialization fails:  %s", RtErrString (rta -> rta_reason));

    sd = rts -> rts_sd;
/*  do authentication using rts -> rts_data here... */
    RTSFREE (rts);

/* read command line arguments here... */
                                                                     30
/* could use RTS_BUSY, RTS_VALIDATE, or RTS_MODE instead and then exit */

    if (RtBeginResponse (sd, RTS_ACCEPT, NULLPE, rti) == NOTOK)
        error ("RT-BEGIN.RESPONSE: %s", RtErrString (rta -> rta_reason));

    if (RoSetService (sd, RoPService, roi) == NOTOK)
        error ("RoSetService:  %s", RoErrString (rop -> rop_reason));

    for (;;)
        switch (result = RoWaitRequest (sd, NOTOK, roi)) {           40
            case NOTOK:
            case OK:
            case DONE:
                ros_indication (sd, roi);
                break;

            default:
                error ("unknown return from RoWaitRequest=%d", result);
        }
}                                                                    50
```

Figure A.5: Initializing the generic RTS responder

```
...

static int  ros_end (sd, roe)
int     sd;
struct RoSAPend *roe;
{
    struct RtSAPindication  rtis;
    register struct RtSAPindication *rti = &rtis;
    register struct RtSAPabort   *rta = &rti -> rti_abort;
                                                                        10
    if (RtEndResponse (sd, rti) == NOTOK)
        error ("RT-END.RESPONSE: %s", RtErrString (rta -> rta_reason));

    exit (0);
}
```

Figure A.6: Finalizing the generic RTS responder

Bibliography

[CCI84a] Message Handling Systems: Presentation Transfer Syntax and Notation. International Telegraph and Telephone Consultative Committee, October 1984. Recommendation X.409.

[CCI84b] Message Handling Systems: Remote Operations and Reliable Transfer Server. International Telegraph and Telephone Consultative Committee, October 1984. Recommendation X.410.

[CCI88a] Reliable Transfer: Model and Service Definition. International Telegraph and Telephone Consultative Committee, March 1988. Recommendation X.218.

[CCI88b] Reliable Transfer: Protocol Specification. International Telegraph and Telephone Consultative Committee, March 1988. Recommendation X.228.

[CCI88c] Remote Operations: Model, Notation and Service Definition. International Telegraph and Telephone Consultative Committee, March 1988. Recommendation X.219.

[CCI88d] Remote Operations: Protocol Specification. International Telegraph and Telephone Consultative Committee, March 1988. Recommendation X.229.

[CCI88e] Specification of Abstract Syntax Notation One. International Telegraph and Telephone Consultative Committee, 1988. Recommendation X.208.

[ECM85] Remote Operations: Concepts, Notation and Connection-Oriented Mappings, December 1985. ECMA TR/31.

[ISO87a] Information Processing — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1). International Organization for Standardization and International Electrotechnical Committee, 1987. International Standard 8824.

[ISO87b] Information Processing — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). International Organization for Standardization and International Electrotechnical Committee, 1987. International Standard 8825.

[ISO88a] Information Processing Systems — Open Systems Interconnection — Service Definition for the Association Control Service Element. International Organization for Standardization and International Electrotechnical Committee, April 1988. Revised Final Text of Draft International Standard 8649.

[ISO88b] Information Processing Systems — Open Systems Interconnection – Protocol Specification for the Association Control Service Element. International Organization for Standardization and International Electrotechnical Committee, April 1988. Revised Final Text of Draft International Standard 8650.

[ISO88c] Information Processing Systems — Text Communication — Reliable Transfer Part 1: Model and Service Definition. International Organization for Standardization and International Electrotechnical Committee, March 1988. Working Document for International Standard 9066-1.

[ISO88d] Information Processing Systems — Text Communication — Reliable Transfer Part 2: Protocol Specification. International Organization for Standardization and International Electrotechnical Committee, March 1988. Working Document for International Standard 9066-2.

[ISO88e] Information Processing Systems — Text Communication — Remote Operations Part 1: Model, Notation and Service Definition. International Organization for Standardization and International