Introduction

NAME

OSS – Open Systems Interconnection Services

SYNOPSIS

#include <oss.h>

All calls, structs and variables in this interface are prefixed with s_ and all values with S_. They are defined in the above include file.

DESCRIPTION

This document specifies a general session-oriented communication interface for programs written in the C language. The interface is based on the internationally standardized Session Service of the ISO Reference Model for Open Systems Interconnection (ISO 8326). A knowledge of the ISO standard is essential for an understanding of this specification. Some terms and characteristics are explained below.

Session applications:

The users of this interface are called ’session applications’ and the interface provider the ’session service’. A session application may consist of one or more processes, and a process may participate in one or more session applications. Session connections are established between two session applications. One is known as the ’local application’ and the other, the partner, as the ’remote application’, even if the local and remote applications reside in the same system. The addressable unit is the session application, which is mapped 1:1 onto a transport application. The address of a session application is thus also the address of the corresponding transport application.

Session connections:

A session connection may be established between a local and a remote session application. A session application may maintain more than one session connection at a given time and more than one connection may exist between the same pair of session applications. A session connection is always tied to one process of the application and at a given time only known in this process. A session connection initiated by a local application is implicitly bound to the process issuing the connect request call. A session connection initiated by a remote application is implicitly bound to the first (or oldest) process of the addressed application. A local function was introduced to explicitly change the association of a connection from one process of the application to another.

Session call techniques:

This interface closely follows the ISO standard and the service primitives of the standard appear here as function calls. Since the standard is an abstract definition covering only the interaction with the remote partner, some local functions have been added to provide a complete programming interface to the session service as a subsystem in an operating system environment.

The service primitives of the standard are of two kinds, requests and responses directed from the user to the provider, and indications and confirmations directed from the provider to the user. Since indications and confirmations may occur at any time unpredictably, a local function s_event was introduced to wait or periodically check for any type of indication or confirmation. The s_event function only announces the occurrence of session events that need to be received immediately with the appropriate event-specific function call. The call receiving the announced indication or confirmation then syntactically resembles the requests and responses.

Parameters to be supplied by the user are marked with ’(⟶)’ and parameters with values to be returned by OSS are marked with ’(⟵)’.

User Interface of OSS V3.0

Differences between the OSS V2.0 and OSS V3.0 Interfaces

In OSS V3.0, the user interface has remained unchanged in comparison with OSS V2.0. However, the following changes have been made in the implementation:

- Maximum data length
  - In OSS V3.0, the maximum data length of the SIDU (session interface data unit) is independent of the maximum length of the TIDU (transport interface data unit). The maximum SIDU length is approx. 64 Kb; this value is returned when s_info() is called.
  - With the exception of s_datarq() and s_typerq(), the following is valid for all service calls: If the version 2 session protocol is used, the maximum user data length increases from 8 Kbytes to 10 Kbytes.
  - In the case of s_datarq() and s_typerq(), the following is valid: the maximum length of the user data is unlimited if the data is linked with S_MORE. However, only one data block per request can be transferred; the maximum permissible length for this data block is the maximum SIDU length (see above).
- The ’Local Functions’ have been extended to include the functions ’s_stop’ and ’s_go’.
- There is a new diagnostic routine (OSSD), see page 122.
- The installation steps have been modified, see page 135.
- Some notes for the case of terminating an OSS application by BCLOSE or by BCAM have been omitted, see page 144.
- Uniform layout of the address structure for OSS V3.0 applications, see page 145.
- With the s_attach(), s_conrq(), s_conin(), and s_redin() calls, the user references transferred are no longer tested for uniqueness.
- s_conin() and s_concf() *always* return the session address.
- The session trace evaluation program STEP has been extended by some options. The session references are output with all trace records.

Readme file

Details of any functional changes to the product version described in this manual are provided in a product-specific readme file. The names of readme files on your BS2000 system have the following structure: `SYSDOC.OSS.030.READ-ME.E`. Please contact your system administrator for details of the user ID under which the readme file you are interested in is stored. You can view a readme file using the `/SHOW-FILE` command or an editor, or have it printed out on a standard printer using the following command:

```
PRINT-FILE FILE-NAME=filename,LAYOUT-CONTROL=PARAMETERS(CONTROL-CHARACTERS=EBCDIC)
```

Changes Required to enable an Existing OSS V2.0 Application to Use OSS V3.0

All applications can be taken over on a one-to-one basis.

Local Functions

Overview

The function calls contained in this chapter are:

s_attach	–	session application attach
s_detach	–	session application detach
s_event	–	announce session service event
s_info	–	request session information
s_timer	–	generate time interrupt event
s_wake	–	wake up another session user process
s_error	–	return error diagnostic code
s_redrq	–	redirect session connection
s_redin	–	receive redirected session connection
s_stop	–	stop indication of connection related events
s_go	–	resume indication of connection related events

The local function calls do not form part of the ISO standard, but are necessary to enable a complete programming interface to be provided.

s_attach

NAME

s_attach – session application attach

SYNOPSIS

```
int  s_attach(aref,auref,addr,NULL)
int  *aref;     (←)
int  *auref;    (→)
char *addr;     (→)
```

DESCRIPTION

’S_attach’ attaches the calling process to the session service. ’Aref’ points to a location in which the session service places the local application reference. It must be included in some session service calls to specify the local session application.

’Auref’ points to the application user reference, which is returned by the session service in the s_event call for the announcements S_CONIN and S_REDIN. It may be used by the session application program to distinguish between a number of session applications attached to the session service.
If no application user reference is being used, ’auref’ points to S_NOUREF or may be NULL. In this case the value S_NOUREF is returned for ’uref’ in the S_CONIN and S_REDIN events.

’Addr’ points to the address of the session application. A session application address consists of a session selector and a transport application address.

The first process issuing an s_attach call with this ’addr’ implicitly creates the session application. Each process using the session service must attach itself before it can use further session service calls.

The last parameter is reserved for future extensions.

RETURN VALUE

S_OK	successful, and application implicitly created
S_NOTFIRST	successful, and application already created by another process
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call
S_RETRY	unsuccessful due to internal resource shortage; it is advisable to retry the call later; a diagnostic code is available via the s_error call

APPLICATION USAGE

An attached application is not only known to the local session service but can also be addressed by partner applications throughout the network.

RESTRICTIONS

This OSS version does not support different session applications attached to the same transport application. This means that if two different processes are attached to the same transport application, they must use the same session selector.
OSS, however, is not able to detect the incorrect use of session selectors in the s_attach call, which may have a strange effect on S_CONIN and S_REDIN events.

NOTE

The structure of the session application address is system-dependent (see appendix D).

RELATIONSHIP TO ISO 8326

Local function needed to make the application processes known to the session service and addressable.

s_detach

NAME

s_detach – session application detach

SYNOPSIS

```
int  s_detach(aref)
int  *aref;     (⟶)
```

DESCRIPTION

’S_detach’ detaches the session application referenced by ’aref’ from the calling process. The last process to issue an s_detach call for a ’addr’ given in s_attach implicitly destroys the session application, after which it is no longer addressable. Session connections known in the calling process implicitly undergo disorderly release from the session service.

RETURN VALUE

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

An attached process terminating either normally or abnormally without an s_detach call being issued is implicitly detached from the session service.

RELATIONSHIP TO ISO 8326

Local function needed as a counterpart to s_attach.

s_event

NAME

s_event – announce session service event

SYNOPSIS

```
int  s_event(sref,uref,cmode,udatal)
int  *sref;             (←)
int  *uref;             (←)
int  cmode;             (→)
unsigned *udatal;       (←)
```

DESCRIPTION

All asynchronous session service events (indications, confirmations and local events) for all session connections known to the calling process are announced by ’s_event’ call. The return value of the call indicates the announced event type. ’Sref’ points to a location in which the session service places the local session reference for session-specific events. ’Uref’ is NULL or points to a location in which the session service places the local session user reference for session-specific events or the application user reference for the events S_CONIN and S_REDIN. If ’uref’ is NULL, no user reference is returned. ’Cmode’ specifies the call mode as either

S_WAIT	wait for the next event to occur, or
S_CHECK	check if a session event is present.

’Udatal’ points to a location where the length of the user data belonging to the event is written.

RETURN VALUES

S_NOEVENT	If ’cmode’=S_CHECK, no session event is present. If ’cmode’=S_WAIT, the blocking s_event was interrupted by a signal or an internal action not leading to a session event. If the user does not wish to terminate, the s_event call should be repeated. No ’sref’ and ’uref’ specified.
S_GO	The stop condition due to a flow control shortage has been cleared for this session connection and the stopped call successfully completed. It is now possible to continue with further request or response calls for this session connection.
S_CONIN	session connect indication to be received with an s_conin call; the value returned for ’uref’ is the session application user reference ’auref’ for the session application ’addr’ attached in a previous s_attach call.
S_CONCF	session connect confirmation to be received with an s_concf call

S_RELIN	session release indication to be received with an s_relin call
S_RELCF	session release confirmation to be released with an s_relcf call
S_UABOIN	user-initiated session abort indication to be received with an s_uaboin call
S_PABOIN	provider-initiated session abort indication to be received with an s_paboin call
S_DATAIN	normal data indication to be received with one or a sequence of s_datain calls; all the number of bytes announced in ’udatal’ must, however, be received before another session call can be issued.
S_TKGIN	token give indication to be received with an s_tkgin call
S_TKPIN	token please indication to be received with an s_tkpin call
S_TYPEIN	typed data indication to be received with one or a sequence of s_typein calls; all the number of bytes announced in ’udatal’ must, however, be received before another session call can be issued.
S_CAPIN	capability data indication to be received with an s_capin call
S_CAPCF	capability data confirmation to be received with an s_capcf call
S_MININ	sync minor indication to be received with an s_minin call
S_MINCF	sync minor confirmation to be received with an s_mincf call
S_MAJIN	sync major indication to be received with an s_majin call
S_MAJCF	sync major confirmation to be received with an s_majcf call
S_SYNIN	resynchronize indication to be received with an s_synin call
S_SYNCF	resynchronize confirmation to be received with an s_syncf call
S_STAIN	activity start indication to be received with an s_stain call
S_RESIN	activity resume indication to be received with an s_resin call
S_INTIN	activity interrupt indication to be received with an s_intin call
S_INTCF	activity interrupt confirmation to be received with an s_intcf call
S_DISIN	activity discard indication to be received with an s_disin call
S_DISCF	activity discard confirmation to be received with an s_discf call
S_ENDIN	activity end indication to be received with an s_endin call
S_ENDCF	activity end confirmation to be received with an s_endcf call
S_CTGIN	control give indication to be received with an s_ctgin call
S_UEXCIN	user-initiated exception report indication to be received with an s_uexcin call
S_PEXCIN	provider-initiated exception report indication to be received with an s_pexcin call
S_REDIN	session redirect indication to be received with an s_redin call; the value returned for ’uref’ is the session application user reference ’auref’ for the session application ’addr’ attached in

	a previous s_attach call
S_TIMEINT	time interrupt generated by a local s_timer call; no ’sref’, ’uref’ specified
S_ERROR	call unsuccessful; a diagnostic code is available via the s_error call; no ’sref’, ’uref’ specified.

APPLICATION USAGE

After receiving a session indication or confirmation via s_event the user must call either the corresponding s_...in s_...cf function to receive the announced event or the s_uaborq function to cancel the session connection.

The s_event call with ’cmode’=S_WAIT is the only blocking call in the session interface and hence the central call, at the top of a dispatcher (switch), in an event-driven session application.

Note that the S_NOEVENT return value may, depending on the implementation, be generated as a result of session layer internal actions of no significance for the session user, such as the reception of transport connect indication or confirmation.

RELATIONSHIP TO ISO 8326

A local function needed to announce the occurrence of asynchronous session events in addition to the abstract ISO specification.

s_info

NAME

s_info – request session information

SYNOPSIS

```
int  s_info(sref,maxl,NULL)
int  *sref;              (→)
unsigned *maxl;          (←)
```

DESCRIPTION

’S_info’ requests information about the session connection with the local reference ’sref’. ’Maxl’ points to a location to which the maximum length of one session interface data unit (SIDU) is written.

The third parameter is reserved for future extensions.

The requester of a session must not call s_info until the session has been fully established (s_concf).

RETURN VALUE

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Local function needed to obtain information about implementation-dependent or dynamically changing session characteristics.

s_timer

NAME

s_timer – generate time interrupt event

SYNOPSIS

```
int  s_timer(sec)
unsigned  sec;  (→)
```

DESCRIPTION

’S_timer’ generates a time interrupt event that is announced via the s_event call after ’sec’ seconds. A second s_timer call issued before the first one has expired implicitly cancels the first interrupt. A ’sec’ value equal to 0 does not generate an interrupt; it merely cancels an interrupt that has not yet expired.

APPLICATION USAGE

This call may be used either to wake up a blocking s_event call so that it does not wait for events for ever that may never occur, or to time-supervise events.

RETURN VALUE

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Local function needed for the time supervision of events and blocking session calls, or as a substitute for the alarm function required internally by the session service.

s_wake

NAME

s_wake – wake up a session user process

SYNOPSIS

```
int  s_wake(pid)
int  *pid;      (⟶)
```

DESCRIPTION

’S_wake’ is used to wake up (release) a session user process blocked by an s_event call. If it is waiting in an s_event call, it will immediately return S_NOEVENT. Otherwise an s_wake call has no effect.

APPLICATION USAGE

This call may be used by one session user process to clear a blocking s_event call in another. The process calling s_wake does not have to be attached to the session service. The process being woken, however, must be attached in order to call s_event.

RETURN VALUE

S_OK	successful
S_RETRY	unsuccessful; process to be woken is not attached to the session service, or system error. s_error cannot be called as no error code is set.

RELATIONSHIP TO ISO 8326

Local function needed to ensure the cooperation of session user processes.

s_error

NAME

s_error – return error diagnostic code

SYNOPSIS

```
int  s_error(addinfo)
int  *addinfo;                     (←)
```

DESCRIPTION

’S_error’ supplies an additional diagnostic code after a session call has returned an S_ERROR or S_RETRY value. The returned codes are intended to support the diagnosis of error conditions and should not be interpreted by the calling software. Moreover, the list of possible codes differs from one implementation to another. ’Addinfo’ points to a location in which the session service places an additional value for the error codes S_SYSERR and S_TSERR.

A list of possible diagnostic codes for the error code S_TSERR is contained in the include file cmx.h. Diagnostic codes for the error code S_SYSERR are listed in the appendix.

APPLICATION USAGE

A session application should always save or display the diagnostic code after the return value S_ERROR and after S_RETRY if the failed call is not retried.

RELATIONSHIP TO ISO 8326

Local function needed for the diagnosis of error conditions.

s_redrq

NAME

s_redrq — session redirect request

SYNOPSIS

```
int  s_redrq(sref,pid,userdata)
int  *sref;                    (→)
int  *pid;                     (→)
struct s_udatas *userdata;     (→)
```

DESCRIPTION

’S_redrq’ asks for the session connection with the local reference ’sref’ to be redirected from the calling process to the process of the same session application with the ID pointed to by ’pid’. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the process. If ’len’ is 0, no user data is transferred. The length of the user data must not exceed 12 Kbytes.

After this call, the session connection is no longer known to the calling process.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

This call may only be used to distribute incoming session connections to server processes when new processes are created to serve incoming connections.

s_redrq() may only be called after s_conin(). The process receiving the session connection must already be attached to the same session application as the redirecting process.

The s_conrs() call must be made by the process receiving the session connection with an s_redin() call.

RELATIONSHIP TO ISO 8326

Local function that adds necessary flexibility to more complex multi-process session applications.

s_redin

NAME

s_redin – receive session redirect indication

SYNOPSIS

```
int  s_redin(sref,suref,aref,pid,userdata)
int  *sref;                      (→)
int  *suref;                     (→)
int  *aref;                      (←)
int  *pid;                       (←)
struct s_udatas *userdata        (←)
```

DESCRIPTION

’S_redin’ receives an indication announced via s_event to redirect the session connection with the local reference ’sref’ to the calling process. ’Suref’ points to a location containing a session connection user reference. It may be specified by the session user to distinguish a number of session connections. It is returned in ’uref’ by all s_event calls concerning a particular session connection. If no session connection user reference is being used, ’suref’ points to S_NOUREF or may be NULL. In this case the value S_NOUREF is returned by the s_event call.

’Aref’ points to a location in which the application reference of the local application for which the session connection redirection was announced is returned. ’Pid’ points to a location to which the ID of the process that requested the redirection is written. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced by the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

This call assigns the session connection to the calling process. If the session connection is not wanted, it either has to be released or returned to the requesting process.

RELATIONSHIP TO ISO 8326

Local function needed, together with the session redirect request function call, for multi-process applications.

s_stop

NAME

s_stop – stop indication of connection related events

SYNOPSIS

```
int  s_stop(sref)
int  *sref;                     (→)
```

DESCRIPTION

’s_stop’ can be used to stop the indication of events related to the connection specified by ’sref’. ’Sref’ points to the reference of the session connection.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

This call may be used to stop the indication of connection related events on a session connection with the exception of the event S_PABOIN.

RELATIONSHIP TO ISO 8326

Local function needed for flow control.

s_go

NAME

s_go – resume indication of connection related events

SYNOPSIS

```
int  s_go(sref)
int  *sref;                      (→)
```

DESCRIPTION

’S_go’ can be used to resume the indication of events related to the connection specified by ’sref’. ’Sref’ points to the reference of the session connection.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

This call is used to cancel the effect of an s_stop call.

RELATIONSHIP TO ISO 8326

Local function needed for flow control.

The Kernel Functional Unit

Overview

The kernel functional unit supports the basic session services required to establish a session connection, transfer normal data and release the session connection.

The kernel functional unit comprises the following calls

s_conrq	–	session connect request
s_conin	–	receive session connect indication
s_conrs	–	session connect response
s_concf	–	receive session connect confirm
s_relrq	–	session release request
s_relin	–	receive session release indication
s_relrs	–	session release response
s_relcf	–	receive session release confirm
s_uaborq	–	user-initiated abort request
s_uaboin	–	receive user-initiated abort indication
s_paboin	–	receive provider-initiated abort indication
s_datarq	–	normal data request
s_datain	–	receive normal data indication

s_conrq

NAME

s_conrq – session connect request

SYNOPSIS

```
int  s_conrq(sref,suref,aref,toaddr,ucid,funits,qos,syncp,token,
                                                      userdata)
int  *sref;                     (←)
int  *suref;                    (→)
int  *aref;                     (→)
char *toaddr;                   (→)
struct s_cid *ucid;             (→)
int  *funits;                   (→)
char *qos;                      (→)
long *syncp;                    (→)
char *token;                    (→)
struct s_udatas *userdata;      (→)
```

DESCRIPTION

’S_conrq’ asks for a session connection to be established to the session application (remote or local) named in ’toaddr’. ’Sref’ points to a location in which the session service returns the local session reference identifying this connection. ’Suref’ points to a location containing a session connection user reference. It may be specified by the session user to distinguish a number of session connections. It is returned in ’uref’ by all s_event calls concerning a particular session connection. If no session connection user reference is being used, ’suref’ points to S_NOUREF or may be NULL. In this case the value S_NOUREF is returned by the s_event call.

’Aref’ points to the application reference of the calling application as returned in a previous s_attach call. ’Toaddr’ points to a structure containing the session service address of the called application. ’Ucid’ is NULL if no user connection identification is required or points to an ’s_cid’ struct containing the user connection identification as follows:

```
struct s_cid {                   * layout of connection ID      *
        int  s_luref;            * length of SS-user reference  *
        char s_uref[64];         * calling SS-user reference    *
        int  s_lcomref;          * length of common reference   *
        char s_comref[64];       * common reference             *
        int  s_laddref;          * length of additional ref     *
        char s_addref[4];        * additional reference info    *
             };
```

’Funits’ specifies the functional units proposed for the session as described in the standard. ’Funits’ values are constructed by ORing values from the following list:

S_HDX	half duplex and data token available
S_FDX	full duplex
S_MINOR	minor synchronization and minor sync token avail.
S_MAJOR	major synchronization and major activity token av.
S_RESYNC	resynchronize
S_ACTIVITY	activity management and major activity token avail.
S_NEGRELEASE	negotiated release and release token available
S_CAPABILITY	capability data (implying S_ACTIVITY)
S_EXCEPTIONS	exceptions (implying S_HDX)
S_TYPED	typed data
S_PVERS1	session protocol version 1 is to be used

’Qos’ is NULL (reserved for quality of service specification in future versions). ’Syncp’ is NULL if no sync point is required, or points to the initial sync point number. The value of the latter is an integer in the range 0-999999, or S_NOVALUE if the parameter is not specified. ’Token’ points to the initial token assignment and the value is constructed by ORing values from the following list:

S_T_DATA	data token on accepter side
S_T_MINOR	minor synchronize token on accepter side
S_T_ACTIVITY	major activity token on accepter side
S_T_RELEASE	release token on accepter side
S_TC_DATA	data token on side chosen by accepter
S_TC_MINOR	minor sync. token on side chosen by accepter
S_TC_ACTIVITY	major activity token on side chosen by accepter
S_TC_RELEASE	release token on side chosen by accepter

If a particular token has no value assigned to it, the token remains on the requester side or is not used in the current session. If all tokens in the session have no value, ’token’ may be NULL. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_RETRY	unsuccessful due to internal resource shortage; in this case it is advisable to retry the call later; a diagnostic code is available via the s_error call.

APPLICATION USAGE

The session connection is established when a positive session connect confirmation (s_concf) is received from the responding application. This event is announced by an s_event call. The sref is not passed on to a child process after a fork call in a UNIX environment.

NOTE

The structure of ’toaddr’ is system-dependent (see appendix D).

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-CONNECT request.

s_conin

NAME

s_conin – receive session connect indication

SYNOPSIS

```
int  s_conin(sref,suref,aref,fraddr,ucid,funits,qos,syncp,token,
                                                     userdata)
int  *sref;                     (→)
int  *suref;                    (→)
int  *aref;                     (←)
char *fraddr;                   (←)
struct s_cid *ucid;             (←)
int  *funits;                   (←)
char *qos;                      (←)
long *syncp;                    (←)
char *token;                    (←)
struct s_udatas *userdata;      (←)
```

DESCRIPTION

'S_conin' receives an indication for session connection establishment announced via s_event for the session connection with the local reference 'sref'. 'Suref' points to a location containing a session connection user reference. It may be specified by the session user to distinguish a number of session connections. It is returned in 'uref' by all s_event calls concerning a particular session connection. If no session connection user reference is being used, 'suref' points to S_NOUREF or may be NULL. In this case the value S_NOUREF is returned by the s_event call.

'Aref' points to a location to which the application reference of the called application as returned in a previous s_attach call is written. 'Fraddr' points to an area to which the session service address of the calling application is written. 'Ucid' points to an 's_cid' struct to which the user connection identifier specified by the partner is written as follows:

```
struct s_cid {                  * layout of connection ID      *
        int  s_luref;           * length of SS-user reference  *
        char s_uref[64];        * calling SS-user reference    *
        int  s_lcomref;         * length of common reference   *
        char s_comref[64];      * common reference             *
        int  s_laddref;         * length of additional ref     *
        char s_addref[4];       * additional reference info    *
             };
```

’Funits’ points to a location to which the functional units proposed by the partner are written. ’Funits’ values are constructed by ORing values from the following list:

S_HDX	half duplex and data token available
S_FDX	full duplex
S_MINOR	minor synchronization and minor sync token avail.
S_MAJOR	major synchronization and major activity token av.
S_RESYNC	resynchronize
S_ACTIVITY	activity management and major activity token avail.
S_NEGRELEASE	negotiated release and release token available
S_CAPABILITY	capability data (implying S_ACTIVITY)
S_EXCEPTIONS	exceptions (implying S_HDX)
S_TYPED	typed data
S_PVERS1	session protocol version 1 is to be used

’Qos’ is NULL (reserved for quality of service specification in future versions).

’Syncp’ points to a location to which the initial sync point number is written. The sync point is an integer in the range 0-999999. If the partner has not specified an initial sync point number, the parameter is set to S_NOVALUE. ’Token’ points to a location to which the initial token assignment is written. The value is constructed by ORing values from the following list:

S_T_DATA	data token on accepter side
S_T_MINOR	minor synchronize token on accepter side
S_T_ACTIVITY	major activity token on accepter side
S_T_RELEASE	release token on accepter side
S_TC_DATA	data token on side chosen by accepter
S_TC_MINOR	minor sync token on side chosen by accepter
S_TC_ACTIVITY	major activity token on side chosen by accepter
S_TC_RELEASE	release token on side chosen by accepter

If a particular token has no value assigned to it, the token remains on the requester side or is not used in the current session. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and has the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

The session connect indication must be answered to with a session connect response call (s_conrs) either accepting or rejecting the connection.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-CONNECT indication.

s_conrs

NAME

s_conrs – session connect response

SYNOPSIS

```
int  s_conrs(sref,aref,ucid,result,funits,qos,syncp,token,userdata)
int  *sref;                     (→)
int  *aref;                     (→)
struct s_cid *ucid;             (→)
char *result;                   (→)
int  *funits;                   (→)
char *qos;                      (→)
long *syncp;                    (→)
char *token;                    (→)
struct s_udatas *userdata;      (→)
```

DESCRIPTION

’S_conrs’ responds to the session connect indication received via s_conin for the session connection with the local reference ’sref’. ’Aref’ points to the application reference of the responding application as returned in a previous s_attach call. The ’result’ of the response is one of the following:

S_ACCEPT	session connect indication accepted
S_REJECT	session connect indication rejected
S_CONGEST	session connect indication rejected due to temporary congestion (no user data is permitted).

’Ucid’ is NULL if no user connection identification is required or points to an ’s_cid’ struct containing the user connection identification as follows:

```
struct s_cid {                 * layout of connection ID      *
        int  s_luref;          * length of SS-user reference  *
        char s_uref[64];       * called SS-user reference     *
        int  s_lcomref;        * length of common reference   *
        char s_comref[64];     * common reference             *
        int  s_laddref;        * length of additional ref     *
        char s_addref[4];      * additional reference info    *
             };
```

’Funits’ specifies the functional units proposed by the responder. ’Funits’ values are constructed by ORing values from the following list:

S_HDX	half duplex and data token available
S_FDX	full duplex (not together with S_HDX)
S_MINOR	minor synchronization and minor sync token avail.
S_MAJOR	major synchronization and major activity token av.
S_RESYNC	resynchronize
S_ACTIVITY	activity management and major activity token avail.
S_NEGRELEASE	negotiated release and release token available
S_CAPABILITY	capability data (implying S_ACTIVITY)
S_EXCEPTIONS	exceptions (implying S_HDX)
S_TYPED	typed data
S_PVERS1	session protocol version 1 has to be used

’Qos’ is NULL (reserved for quality of service specification in future versions).

’Syncp’ is NULL if no sync point is required, or points to the initial sync point number. The value of the latter is an integer in the range 0-999999 or S_NOVALUE if the parameter is not specified. ’Token’ specifies the tokens chosen or requested by the responder and the value is constructed by ORing values from the following list:

S_T_DATA	data token on accepter side
S_T_MINOR	minor synchronize token on accepter side
S_T_ACTIVITY	major activity token on accepter side
S_T_RELEASE	release token on accepter side

If a particular token has no value assigned to it, the token remains on the requester side or is not used in the current session. If tokens assigned to the requester are specified, an S-TOKEN-PLEASE indication is implicitly generated on the requester side after the S-CONNECT confirm. If all tokens in the session have no value, ’token’ may be NULL. ’Userdata’ is NULL if result is S_CONGEST or no user data is required or it points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session. This value is only possible if the result is S_ACCEPT.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-CONNECT response.

s_concf

NAME

s_concf – receive session connect confirmation

SYNOPSIS

```
int  s_concf(sref,toaddr,ucid,result,funits,qos,syncp,token,userdata)
int  *sref;                     (→)
char *toaddr;                   (←)
struct s_cid *ucid;             (←)
char *result;                   (←)
int  *funits;                   (←)
char *qos;                      (←)
long *syncp;                    (←)
char *token;                    (←)
struct s_udatas *userdata;      (←)
```

DESCRIPTION

’S_concf’ receives the session connect confirmation announced via s_event for the session connection with the local reference ’sref’, in response to a previously issued session connect request call. ’Toaddr’ points to a structure to which the session service address of the responding application is written. ’Result’ points to a location in which the response to the request is placed. Possible responses are:

S_ACCEPT	connection accepted
S_REJECT	connection rejected by partner
S_CONGEST	connection rejected by partner due to temporary congestion
S_PREJECT	connection rejected by session service
S_PCONGEST	connection rejected by session service due to temporary congestion
S_PUNKNOWN	connection rejected by session service since the called session application is unknown
S_PNATTACH	connection rejected by session service since the called session application is not attached
S_PPVERS	connection rejected by session service since the proposed protocol version is not supported

’Ucid’ points to an ’s_cid’ struct to which the user connection identifier specified by the partner is written as follows:

```
struct s_cid {                  * layout of connection ID      *
        int  s_luref;           * length of SS-user reference  *
        char s_uref[64];        * called SS-user reference     *
        int  s_lcomref;         * length of common reference   *
        char s_comref[64];      * common reference             *

        int  s_laddref;         * length of additional ref     *
        char s_addref[4];       * additional reference info    *
             };
```

’Funits’ points to a location to which the functional units proposed by the partner are written. ’Funits’ values are constructed by ORing values from the following list:

S_HDX	half duplex and data token available
S_FDX	full duplex (not together with S_HDX)
S_MINOR	minor synchronization and minor sync token avail.
S_MAJOR	major synchronization and major activity token av.
S_RESYNC	resynchronize
S_ACTIVITY	activity management and major activity token avail.
S_NEGRELEASE	negotiated release and release token available
S_CAPABILITY	capability data (implying S_ACTIVITY)
S_EXCEPTIONS	exceptions (implying S_HDX)
S_TYPED	typed data
S_PVERS1	session protocol version 1 is to be used

’Qos’ is NULL (reserved for quality of service specification in future versions).

’Syncp’ points to a location to which the initial sync point number is written. If the partner has not specified a sync point, the parameter is set to S_NOVALUE. ’Token’ points to a location in which the tokens chosen by the responder are placed. The token value is constructed by ORing values from the following list:

S_T_DATA	data token on accepter side
S_T_MINOR	minor synchronize token on accepter side
S_T_ACTIVITY	major activity token on accepter side
S_T_RELEASE	release token on accepter side

If a particular token has no value, either the token assignment has already been specified by the requester or, if the accepter was given the choice, the token assignment is on the requester side.

’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner are written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-CONNECT confirm.

s_relrq

NAME

s_relrq – session release request

SYNOPSIS

```
int  s_relrq(sref,userdata)
int  *sref;                      (→)
struct s_udatas *userdata;       (→)
```

DESCRIPTION

’S_relrq’ asks for an orderly release of the established session connection with the local reference ’sref’. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

The session connection is released when a positive session release confirmation (s_relcf) is received from the partner application. This event is announced by an s_event call. The s_relrq call is subject to the token restrictions in appendix A.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-RELEASE request.

s_relin

NAME

s_relin – receive session release indication

SYNOPSIS

```
int  s_relin(sref,userdata)
int  *sref;                     (→)
struct s_udatas *userdata       (←)
```

DESCRIPTION

’S_relin’ receives an indication announced via s_event to release the session connection with the local reference ’sref’. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

The session release indication must be answered with a positive or negative session release response call (s_relrs).

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-RELEASE indication.

s_relrs

NAME

s_relrs – session release response

SYNOPSIS

```
int  s_relrs(sref,result,userdata)
int  *sref;                      (→)
char *result;                    (→)
struct s_udatas *userdata;       (→)
```

DESCRIPTION

’S_relrs’ responds to the session release indication received via s_relin for the session connection with the local reference ’sref’. The ’result’ of the response may be either

S_AFFIRMATIVE	response positive and connection released, or
S_NEGATIVE	response negative and connection not released. (Only with the negotiated release functional unit.)

’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session. This value is only possible if result is S_NEGATIVE.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-RELEASE response.

s_relcf

NAME

s_relcf – receive session release confirm

SYNOPSIS

```
int  s_relcf(sref,result,userdata)
int  *sref;                       (→)
char *result;                     (←)
struct s_udatas *userdata         (←)
```

DESCRIPTION

’S_relcf’ receives a session release confirmation announced via s_event for the session connection with the local reference ’sref’, in response to a previously issued session release request call. The ’result’ may be either

S_AFFIRMATIVE	response positive and connection released, or
S_NEGATIVE	response negative and connection not released.

’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-RELEASE confirm.

s_uaborq

NAME

s_uaborq – user-initiated session abort request

SYNOPSIS

```
int  s_uaborq(sref,userdata)
int  *sref;                      (→)
struct s_udatas *userdata;       (→)
```

DESCRIPTION

'S_uaborq' requests a disorderly release of the established session connection with the local reference 'sref'. 'Userdata' is NULL if no user data is required, or points to an 's_udatas' struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

'Ptr' points to an area with 'len' bytes of user data to be transferred to the partner. If 'len' is 0, no user data is transferred. 'Len' is limited to 9 if session protocol version 1 was negotiated.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

This call is the only request or response permitted in a stop condition due to flow control shortage (S_STOP), or in the event of an outstanding s_datarq s_typerq with 'chain'=S_END or an outstanding indication or confirmation call.

This call releases the session connection immediately and any data in transit is lost.

NOTE

The user data specified in this call may be lost, depending on the state of the underlying transport connection.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-U-ABORT request.

s_uaboin

NAME

s_uaboin – receive user-initiated session abort indication

SYNOPSIS

```
int  s_uaboin(sref,userdata)
int  *sref;                     (→)
struct s_udatas *userdata       (←)
```

DESCRIPTION

’S_uaboin’ receives an indication announced via s_event to release the session connection with the local reference ’sref’ abnormally. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

This call releases the session immediately. Any data in transit is lost.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-U-ABORT indication.

s_paboin

NAME

s_paboin – receive provider-initiated session abort indication

SYNOPSIS

```
int  s_paboin(sref,reason)
int  *sref;     (⟶)
int  *reason;   (⟵)
```

DESCRIPTION

’S_paboin’ receives an abnormal release announced via s_event and initiated by the provider for the session connection with the local reference ’sref’. ’Reason’ indicates the abort reason, which may be any of the following:

S_NOREASON	no reason specified
S_TCDISCON	transport connection cleared
S_PROTERROR	session protocol error

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

This call releases the session immediately. Any data in transit is lost.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-P-ABORT indication.

s_datarq

NAME

s_datarq – normal data request

SYNOPSIS

```
int  s_datarq(sref,ptr,len,chain)
int  *sref;     (→)
char *ptr;      (→)
unsigned *len;  (→)
char *chain;    (→)
```

DESCRIPTION

’S_datarq’ requests ’len’ bytes of normal user data from the area pointed to by ’ptr’ to be sent over the session connection with the local reference ’sref’. ’Chain’ specifies if this session interface data unit (SIDU) concludes a session service data unit (SSDU) or not, and if concatenation is to be used, with one of the following values:

S_MORE	This SIDU is not the end of an SSDU.
S_END	This SIDU concludes an SSDU.
S_CONCAT	This SIDU concludes an SSDU and is immediately followed by a session call to be concatenated with this call. (Rules for concatenation in appendix C.)

The SSDU is the unit of data exchanged between two session applications. The SIDU is the data unit exchanged at the local interface. The maximum length of an SIDU is implementation-dependent and can be queried using the s_info call.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

The ’chain’ parameter is useful on the one hand for segmenting an SSDU too big to fit into one SIDU and on the other for combining smaller portions of data (e.g. protocol headers) to form an SSDU.

NOTE

If an application has sent an SIDU with the chain parameter set to S_MORE, it is not allowed to issue any session request or response call except s_uaborq until the SSDU is completed.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-DATA request.

s_datain

NAME

s_datain – receive normal data indication

SYNOPSIS

```
int  s_datain(sref,ptr,len,chain)
int  *sref;      (→)
char *ptr;       (←)
unsigned *len;   (→)
char *chain;     (←)
```

DESCRIPTION

’S_datain’ receives a session interface data unit (SIDU) announced via s_event of normal user data for the session connection with the local reference ’sref’. ’Ptr’ points to an area of ’len’ bytes to which the user data is written. If ’len’ is less than the length announced via s_event, the rest of the data must be received in one or a sequence of s_datain calls until all the announced data has been received, before any further session calls can be issued. ’Chain’ points to a location in which the session service indicates if the received SIDU concludes a session service data unit (SSDU) or not, with either

S_MORE	This SIDU is not the end of an SSDU, or
S_END	This SIDU concludes an SSDU.

The SSDU is the unit of data exchanged between two session applications. The s_event always announces one SIDU, a data unit that is only meaningful at the local interface and has an implementation-dependent maximum size. If ’ptr’ is NULL, ’len’ bytes are discarded by the session service and not delivered to the application.

RETURN VALUES

>0	number of bytes still to be received in the announced SIDU
S_OK	one complete SIDU successfully received
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

Using the ’len’ parameter, an announced SIDU may be received in smaller segments with a sequence of s_datain calls. If the chain indicator has been set to S_MORE and a session indication not equal to S_DATAIN is announced to the application, the end of the SSDU has been discarded and can no longer be given to the application.

NOTE

Even if the chain indicator is set to S_MORE, there is no minimum size of SIDU the user can be sure of receiving.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-DATA indication.

The Half-Duplex Functional Unit

Overview

The half-duplex functional unit supports the half-duplex service. The data token is available when this functional unit is selected.

The half-duplex functional unit comprises the following function calls:

s_tkgrq	–	token give request
s_tkgin	–	receive token give indication
s_tkprq	–	token please request
s_tkpin	–	receive token please indication

s_tkgrq

NAME

s_tkgrq – token give request

SYNOPSIS

```
int  s_tkgrq(sref,token,userdata)
int  *sref;                      (→)
char *token;                     (→)
struct s_udatas *userdata;       (→)
```

DESCRIPTION

’S_tkgrq’ asks for tokens specified in ’token’ for the session connection with the local reference ’sref’ to be passed to the partner session application. ’Token’ points to a value constructed by ORing token values from the following list:

S_T_DATA	data token on partner side
S_T_MINOR	minor synchronize token on partner side
S_T_ACTIVITY	major activity token on partner side
S_T_RELEASE	release token on partner side

’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. If session protocol version 1 was negotiated, no user data is permitted.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-TOKEN-GIVE request.

s_tkgin

NAME

s_tkgin – receive token give indication

SYNOPSIS

```
int  s_tkgin(sref,token,userdata)
int  *sref;                     (→)
char *token;                    (←)
struct s_udatas *userdata;      (←)
```

DESCRIPTION

’S_tkgin’ receives a token give indication announced via s_event for the session connection with the local reference ’sref’. ’Token’ points to a location to which a value is written specifying the tokens that have been passed to this session application. The value is constructed by ORing values from the following list:

S_T_DATA	data token on this side of the session
S_T_MINOR	minor synchronize token on this side
S_T_ACTIVITY	major activity token on this side
S_T_RELEASE	release token on this side

Tokens not involved in the session should be ignored.

’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored. If session protocol version 1 was negotiated, no user data is announced. In this case ’userdata’ may be NULL.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-TOKEN-GIVE indication.

s_tkprq

NAME

s_tkprq – token please request

SYNOPSIS

```
int  s_tkprq(sref,token,userdata)
int  *sref;                       (→)
char *token;                      (→)
struct s_udatas *userdata;        (→)
```

DESCRIPTION

’S_tkprq’ asks for the tokens specified in ’token’ for the session connection with the local reference ’sref’ to be given to the calling session application. ’Token’ points to a value constructed by ORing token values from the following list:

S_T_DATA	data token requested
S_T_MINOR	minor synchronize token requested
S_T_ACTIVITY	major activity token requested
S_T_RELEASE	release token requested

’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-TOKEN-PLEASE request.

s_tkpin

NAME

s_tkpin – receive token please indication

SYNOPSIS

```
int  s_tkpin(sref,token,userdata)
int  *sref;                       (→)
char *token;                      (←)
struct s_udatas *userdata;        (←)
```

DESCRIPTION

’S_tkpin’ receives a token please indication announced via s_event for the session connection with the local reference ’sref’. ’Token’ points to a location to which a value is written specifying the tokens that are wanted from the partner application. The value is constructed by ORing values from the following list:

S_T_DATA	data token requested
S_T_MINOR	minor synchronize token requested
S_T_ACTIVITY	major activity token requested
S_T_RELEASE	release token requested

Tokens not involved in the session should be ignored. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-TOKEN-PLEASE indication.

The Minor Synchronize Functional Unit

Overview

The minor synchronize functional unit supports the minor synchronization point service. The synchronize minor token is available when this functional unit is selected.

The minor synchronize functional unit comprises the following calls:

s_minrq	–	sync minor request
s_minin	–	sync minor indication
s_minrs	–	sync minor response
s_mincf	–	sync minor confirm

s_minrq

NAME

s_minrq – sync minor request

SYNOPSIS

```
int  s_minrq(sref,mtype,syncp,userdata,chain)
int  *sref;                     (→)
char *mtype;                    (→)
long *syncp;                    (←)
struct s_udatas *userdata;      (→)
char chain;                     (→)
```

DESCRIPTION

’S_minrq’ asks for a minor synchronization point to be defined for the session with the local reference ’sref’. ’Mtype’ specifies whether a confirmation is required or not, with either

S_EXPLICIT	A confirmation from the partner is required, or
S_OPTIONAL	No confirmation is required.

’Syncp’ points to a location to which the session service writes the identification number of the sync point. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;  };       * length of user data          *
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. ’Chain’ specifies if this function call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

Data request calls, and even further sync minor request calls, may be issued before a requested confirmation is received. If the activity management functional unit was negotiated, the call can only be issued within an activity. The s_minrq call is subject to the token restrictions in appendix A.

NOTE

It is up to the session user to ensure that the sync point number does not exceed 999998.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-SYNCH-MINOR request.

s_minin

NAME

s_minin – receive sync minor indication

SYNOPSIS

```
int  s_minin(sref,mtype,syncp,userdata)
int  *sref;                      (→)
char *mtype;                     (←)
long *syncp;                     (←)
struct s_udatas *userdata;       (←)
```

DESCRIPTION

’S_minin’ receives an indication announced via s_event to define a minor synchronization point for the session connection with the local reference ’sref’. ’Mtype’ points to a location to which the sync point type is written, as either

S_EXPLICIT	A response to the sync point is required, or
S_OPTIONAL	No response is required.

’Syncp’ points to a location to which the identification number of the sync point is written. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-SYNCH-MINOR indication.

s_minrs

NAME

s_minrs – sync minor response

SYNOPSIS

```
int  s_minrs(sref,syncp,userdata,chain)
int  *sref;                       (→)
long *syncp;                      (→)
struct s_udatas *userdata;        (→)
char chain;                       (→)
```

DESCRIPTION

’S_minrs’ responds to a sync minor indication received via s_minin for the session with the local reference ’sref’. ’Syncp’ points to the identification number of the sync point. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. ’Chain’ specifies if this function call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-SYNCH-MINOR response.

s_mincf

NAME

s_mincf – receive sync minor confirm

SYNOPSIS

```
int  s_mincf(sref,syncp,userdata)
int  *sref;                       (→)
long *syncp;                      (←)
struct s_udatas *userdata;        (←)
```

DESCRIPTION

’S_mincf’ receives a sync minor confirm announced via s_event for the session connection with the local reference ’sref’, in response to a previously given sync minor request. ’Syncp’ points to a location to which the identification number of the sync point is written. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-SYNCH-MINOR confirm.

The Activity Management Functional Unit

Overview

The activity management functional unit supports the activity management services and the give control service. The major activity token is available when this functional unit is selected.

The activity management functional unit comprises the following calls:

s_starq	–	activity start request
s_stain	–	activity start indication
s_resrq	–	activity resume request
s_resin	–	activity resume indication
s_intrq	–	activity interrupt request
s_intin	–	activity interrupt indication
s_intrs	–	activity interrupt response
s_intcf	–	activity interrupt confirm
s_disrq	–	activity discard request
s_disin	–	activity discard indication
s_disrs	–	activity discard response
s_discf	–	activity discard confirm
s_endrq	–	activity end request
s_endin	–	activity end indication
s_endrs	–	activity end response
s_endcf	–	activity end confirm
s_ctgrq	–	control give request
s_ctgin	–	control give indication

s_starq

NAME

s_starq – activity start request

SYNOPSIS

```
int  s_starq(sref,uactid,userdata,chain);
int  *sref;                     (→)
struct s_aid    *uactid;        (→)
struct s_udatas *userdata;      (→)
char chain;                     (→)
```

DESCRIPTION

’S_starq’ asks for a new activity to be initiated for the session connection with the local reference ’sref’. ’Uactid’ points to an ’s_aid’ struct containing the user activity identifier as follows:

```
struct s_aid {                 * layout of activity ID        *
         int  s_lactid;        * length of ID (min 1, max 6)  *
         char s_actid[6];      * activity identifier, trans-  *
               };              * parent to session service    *
```

’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;        * pointer to user data area    *
        unsigned  len;         * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. ’Chain’ specifies if this call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

The call can only be initiated if no activity is in progress and is subject to the token restrictions in appendix A.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-START request.

s_stain

NAME

s_stain – receive activity start indication

SYNOPSIS

```
int  s_stain(sref,uactid,userdata)
int  *sref;                       (→)
struct s_aid    *uactid;          (←)
struct s_udatas *userdata;        (←)
```

DESCRIPTION

’S_stain’ receives an indication announced via s_event for a new activity to be initiated for the session connection with the local reference ’sref’. ’Uactid’ points to an ’s_aid’ struct to which the user activity identifier is written as follows:

```
struct s_aid   {                * layout of activity ID        *
         int  s_lactid;         * length of ID (min 1, max 6)  *
         char s_actid[6];       * activity identifier, trans-  *
               };               * parent to session service    *
```

’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;         * pointer to user data area    *
        unsigned  len;          * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-START indication.

s_resrq

NAME

s_resrq – activity resume request

SYNOPSIS

```
int  s_resrq(sref,uactid,oldactid,syncp,oldcid,userdata,chain)
int  *sref;                     (→)
struct s_aid    *uactid;        (→)
struct s_aid    *oldactid;      (→)
long *syncp;                    (→)
struct s_ocid   *oldcid;        (→)
struct s_udatas *userdata;      (→)
char chain;                     (→)
```

DESCRIPTION

’S_resrq’ asks for a previously interrupted activity to be resumed on the session connection with the local reference ’sref’. ’Uactid’ points to an ’s_aid’ struct containing the user activity identifier as follows:

```
struct s_aid {                  * layout of activity ID        *
         int  s_lactid;         * length of ID (min 1, max 6)  *
         char s_actid[6];       * activity identifier, trans-  *
               };               * parent to session service    *
```

’Oldact’ points to an ’s_aid’ struct containing the original identifier of the activity being resumed. ’Syncp’ points to the sync point number at which the activity is to be resumed. ’Oldcid’ is NULL or points to an ’s_ocid’ struct containing the identifier for the session connection on which the activity was started, as follows:

```
struct s_ocid {                 * layout of connection ID      *
        int  s_lcguref;         * length of SS-user reference  *
        char s_cguref[64];      * calling SS-user reference    *
        int  s_lcomref;         * length of common reference   *
        char s_comref[64];      * common reference             *
        int  s_laddref;         * length of additional ref     *
        char s_addref[4];       * additional reference info    *
        int  s_lcduref;         * length of SS-user reference  *
        char s_cduref[64];      * called SS-user reference     *
              };
```

’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. ’Chain’ specifies if this session service call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

This call can only be initiated if no activity is in progress and is subject to the token restrictions in appendix A.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-RESUME request.

s_resin

NAME

s_resin – activity resume indication

SYNOPSIS

```
int  s_resin(sref,uactid,oldactid,syncp,oldcid,userdata)
int  *sref;                     (⟶)
struct s_aid    *uactid;        (⟵)
struct s_aid    *oldactid;      (⟵)
long *syncp;                    (⟵)
struct s_ocid   *oldcid;        (⟵)
struct s_udatas *userdata;      (⟵)
```

DESCRIPTION

’S_resin’ receives an indication announced via s_event for a previously interrupted activity to be resumed on the session connection with the local reference ’sref’. ’Uactid’ points to an ’s_aid’ struct to which the user activity identifier is written as follows:

```
struct s_aid {                 * layout of activity ID        *
         int  s_lactid;        * length of ID (min 1, max 6)  *
         char s_actid[6];      * activity identifier, trans-  *
               };              * parent to session service    *
```

’Oldact’ points to an ’s_aid’ struct to which the original identifier of the activity being resumed is written. ’Syncp’ points to a location to which the sync point number is written, at which the interrupted activity is to be resumed. ’Oldcid’ points to an ’s_ocid’ struct to which the identifier of the session connection on which the activity was started, is written as follows:

```
struct s_ocid {                * layout of connection ID      *
        int  s_lcguref;        * length of SS-user reference  *
        char s_cguref[64];     * calling SS-user reference    *
        int  s_lcomref;        * length of common reference   *
        char s_comref[64];     * common reference             *
        int  s_laddref;        * length of additional ref     *
        char s_addref[4];      * additional reference info    *
        int  s_lcduref;        * length of SS-user reference  *
        char s_cduref[64];     * called SS-user reference     *
              };
```

If the partner has not specified an old session identifier, all length parameters are set to 0 by the session service. If no old session identifier is expected, ’oldcid’ may be NULL. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-RESUME indication.

s_intrq

NAME

s_intrq – activity interrupt request

SYNOPSIS

```
int  s_intrq(sref,reason,userdata)
int  *sref;                     (⟶)
int  *reason;                   (⟶)
struct s_udatas *userdata;      (⟶)
```

DESCRIPTION

’S_intrq’ requests the interruption of the current activity on the session connection with the local reference ’sref’. An interrupted activity can be resumed later with the s_resrq call. ’Reason’ is NULL or points to the interrupt reason, which may be any of the following:

S_NOREASON	no specific reason
S_OVERLOAD	user receiving ability jeopardized
S_SEQERR	user sequence error
S_LOCALERR	local application error
S_PROCERR	unrecoverable procedural error
S_DATATOKEN	demand data token

’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. If session protocol version 1 was negotiated, no user data is permitted.

This call may result in the loss of undelivered data.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

The activity is interrupted when an activity interrupt confirm is received from the responding application. This event is announced via the s_event call. The s_intrq call is subject to the token restrictions in appendix A.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-INTERRUPT request.

s_intin

NAME

s_intin – activity interrupt indication

SYNOPSIS

```
int  s_intin(sref,reason,userdata)
int *sref;                          (→)
int *reason;                        (←)
struct s_udatas *userdata;          (←)
```

DESCRIPTION

’S_intin’ receives the activity interrupt indication announced via s_event for the session connection with the local reference ’sref’. ’Reason’ points to a location to which the reason for the interruption is written, which may be any of the following

S_NOREASON	no specific reason
S_OVERLOAD	user receiving ability jeopardized
S_SEQERR	user sequence error
S_LOCALERR	local application error
S_PROCERR	unrecoverable procedural error
S_DATATOKEN	demand data token

’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored. If session protocol version 1 was negotiated, no user data is announced. In this case ’userdata’ may be NULL.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

The activity interrupt indication must be answered with an activity interrupt response call (s_intrs).

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-INTERRUPT indication.

s_intrs

NAME

s_intrs – activity interrupt response

SYNOPSIS

```
int  s_intrs(sref,userdata,chain)
int  *sref;                      (→)
struct s_udatas *userdata;       (→)
char chain;                      (→)
```

DESCRIPTION

’S_intrs’ supplies a response to the activity interrupt indication received via s_intin for the session connection with the local reference ’sref’. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. If session protocol version 1 was negotiated, no user data is permitted. ’Chain’ specifies if this session service call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

Once this response has been issued, no tokens are assigned to the local application.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-INTERRUPT response.

s_intcf

NAME

s_intcf – activity interrupt confirm

SYNOPSIS

```
int  s_intcf(sref,userdata)
int  *sref;                          (→)
struct s_udatas *userdata;           (←)
```

DESCRIPTION

’S_intcf’ receives an activity interrupt confirm announced via s_event for the session connection with the local reference ’sref’, in response to a previously issued activity interrupt request.

’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored. If session protocol version 1 was negotiated, no user data is announced. In this case ’userdata’ may be NULL.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

Upon receipt of this confirmation, all available tokens are assigned to this application.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-INTERRUPT confirm.

s_disrq

NAME

s_disrq – activity discard request

SYNOPSIS

```
int  s_disrq(sref,reason,userdata)
int  *sref;                     (→)
int  *reason;                   (→)
struct s_udatas *userdata;      (→)
```

DESCRIPTION

’S_disrq’ requests abnormal termination of the current activity for the session connection with the local reference ’sref’. ’Reason’ is NULL or points to the discard reason, which may be one of the following:

S_NOREASON	no specific reason
S_OVERLOAD	user receiving ability jeopardized
S_SEQERR	user sequence error
S_LOCALERR	local application error
S_PROCERR	unrecoverable procedural error
S_DATATOKEN	demand data token

’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. If session protocol version 1 was negotiated, no user data is permitted.

This call may result in the loss of undelivered data.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

The activity is discarded when an activity discard confirm (s_discf) is received from the partner application. This event is announced via the s_event call. The s_disrq call is subject to the token restrictions in appendix A.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-DISCARD request.

s_disin

NAME

s_disin – activity discard indication

SYNOPSIS

```
int  s_disin(sref,reason,userdata)
int *sref;                          (→)
int *reason;                        (←)
struct s_udatas *userdata;          (←)
```

DESCRIPTION

’S_disin’ receives an activity discard indication announced via s_event for the session connection with the local reference ’sref’. ’Reason’ points to a location to which the reason for the discard is written, which may be any of the following:

S_NOREASON	no specific reason
S_OVERLOAD	user receiving ability jeopardized
S_SEQERR	user sequence error
S_LOCALERR	local application error
S_PROCERR	unrecoverable procedural error
S_DATATOKEN	demand data token

’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored. If session protocol version 1 was negotiated, no user data is announced. In this case ’userdata’ may be NULL.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

The activity discard indication must be answered with an activity discard response call (s_disrs).

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-DISCARD indication.

s_disrs

NAME

s_disrs – activity discard response

SYNOPSIS

```
int  s_disrs(sref,userdata,chain)
int  *sref;                      (→)
struct s_udatas *userdata;       (→)
char chain;                      (→)
```

DESCRIPTION

’S_disrs’ responds to the activity discard indication received via s_disin for the session connection with the local reference ’sref’. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. If session protocol version 1 was negotiated, no user data is permitted. ’Chain’ specifies if this call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

Once this response has been issued, the local application no longer has any tokens assigned to it.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-DISCARD response.

s_discf

NAME

s_discf – activity discard confirm

SYNOPSIS

```
int  s_discf(sref,userdata)
int  *sref;                     (→)
struct s_udatas *userdata;      (←)
```

DESCRIPTION

’S_discf’ receives an activity discard confirm announced via s_event for the session connection with the local reference ’sref’, in response to a previously issued activity discard request call. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored. If session protocol version 1 was negotiated, no user data is announced. In this case ’userdata’ may be NULL.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

Upon receipt of this confirmation, all available tokens are assigned to this application.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-DISCARD confirmation.

s_endrq

NAME

s_endrq – activity end request

SYNOPSIS

```
int  s_endrq(sref,syncp,userdata,chain)
int  *sref;                    (→)
long *syncp;                   (←)
struct s_udatas *userdata;     (→)
char chain;                    (→)
```

DESCRIPTION

’S_endrq’ requests normal termination of the current activity on the session connection with the local reference ’sref’. ’Syncp’ points to a location to which the sync point number ending the activity is written. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. ’Chain’ specifies if this call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

The activity is terminated when an activity end confirm (s_endcf) is received from the responding application. This event is announced via the s_event call. The s_endrq call is subject to the token restrictions in appendix A.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-END request.

s_endin

NAME

s_endin – activity end indication

SYNOPSIS

```
int  s_endin(sref,syncp,userdata)
int  *sref;                      (⟶)
long *syncp;                     (⟵)
struct s_udatas *userdata;       (⟵)
```

DESCRIPTION

’S_endin’ receives an activity end indication announced via s_event for the session connection with the local reference ’sref’. ’Syncp’ points to a location to which the ending sync point number is written. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

The activity end indication must be answered to with an activity end response call (s_endrs).

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-END indication.

s_endrs

NAME

s_endrs – activity end response

SYNOPSIS

```
int  s_endrs(sref,userdata,chain)
int  *sref;                       (→)
struct s_udatas *userdata;        (→)
char chain;                       (→)
```

DESCRIPTION

’S_endrs’ responds to the activity end indication received via s_endin for the session connection with the local reference ’sref’. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. ’Chain’ specifies if this call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-END response.

s_endcf

NAME

s_endcf – activity end confirm

SYNOPSIS

```
int  s_endcf(sref,userdata)
int  *sref;                        (→)
struct s_udatas *userdata;         (←)
```

DESCRIPTION

’S_endcf’ receives an activity end confirm announced via s_event for the session connection with the local reference ’sref’, in response to a previously issued activity end request. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-ACTIVITY-END confirm.

s_ctgrq

NAME

s_ctgrq – control give request

SYNOPSIS

```
int  s_ctgrq(sref,userdata)
int  *sref;                       (→)
struct s_udatas *userdata;        (→)
```

DESCRIPTION

’S_ctgrq’ asks for the entire set of available tokens to be surrendered for the session connection with the local reference ’sref’. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. If session protocol version 1 was negotiated, no user data is permitted.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

This service can only be requested if the activity functional unit was selected but no activity is in progress. The s_ctgrq call is subject to the token restrictions in appendix A.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-CONTROL-GIVE request.

s_ctgin

NAME

s_ctgin – control give indication

SYNOPSIS

```
int  s_ctgin(sref,userdata)
int  *sref;                     (⟶)
struct s_udatas *userdata;      (⟵)
```

DESCRIPTION

’S_ctgin’ receives a control give indication announced via s_event for the session connection with the local reference ’sref’. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored. If session protocol version 1 was negotiated, no user data is announced. In this case ’userdata’ may be NULL.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-CONTROL-GIVE indication.

The Exceptions Functional Unit

Overview

The exceptions functional unit supports the user and provider exception reporting services.

The exceptions functional unit comprises the function calls:

s_uexcrq	–	user-initiated exception report request
s_uexcin	–	user-initiated exception report indication
s_pexcin	–	provider-initiated exception report indication

s_uexcrq

NAME

s_uexcrq – user-initiated exception report request

SYNOPSIS

```
int  s_uexcrq(sref,reason,userdata,chain)
int *sref;                        (→)
int *reason;                      (→)
struct s_udatas *userdata;        (→)
char chain;                       (→)
```

DESCRIPTION

’S_uexcrq’ asks to report an exceptional condition for the session connection with the local reference ’sref’. ’Reason’ is NULL or points to the exception report reason, which may be any of the following:

S_NOREASON	no specific reason
S_OVERLOAD	user receiving ability jeopardized
S_SEQERR	user sequence error
S_LOCALERR	local application error
S_PROCERR	unrecoverable procedural error
S_DATATOKEN	demand data token

’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. ’Chain’ specifies if this call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

The user exception report can only be used with the half-duplex functional unit. If used with the activity management functional unit, it is only permitted within an activity. The s_uexcrq call is subject the token restrictions in appendix A. After this call, the only call the application is permitted to issue is s_uaborq; all data is discarded until the error situation is cleared.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-U-EXCEPTION-REPORT request.

s_uexcin

NAME

s_uexcin – user-initiated exception report indication

SYNOPSIS

```
int  s_uexcin(sref,reason,userdata)
int *sref;                         (⟶)
int *reason;                       (⟵)
struct s_udatas *userdata;         (⟵)
```

DESCRIPTION

’S_uexcin’ receives a report on an exceptional condition announced via s_event for ’the session connection with the local reference ’sref’. ’Reason’ points to a location to which the reason for the exception report is written, which may be any of the following:

S_NOREASON	no specific reason
S_OVERLOAD	user receiving ability jeopardized
S_SEQERR	user sequence error
S_LOCALERR	local application error
S_PROCERR	unrecoverable procedural error
S_DATATOKEN	demand data token

’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

Upon receipt of this indication the application may only issue the following calls: s_synrq, s_uaborq, s_intrq, s_disrq or s_tkgrq (data token) to clear the error condition. If the application was currently to clear the error condition. If the application was currently sending data with the chain indicator set to S_MORE, the SSDU must be concluded before any further reaction is possible. All data is discarded and no sync point indications are given to the application until the error condition has been cleared.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-U-EXCEPTION-REPORT indication.

s_pexcin

NAME

s_pexcin – provider-initiated exception report indication

SYNOPSIS

```
int  s_pexcin(sref,reason)
int *sref;      (⟶)
int *reason;    (⟵)
```

DESCRIPTION

’S_pexcin’ receives a report announced via s_event on an exceptional condition initiated by the session service, for the session connection with the local reference ’sref’. ’Reason’ points to a location to which the reason for the exception is written, which may be either

S_NOREASON	no specific reason stated, or
S_PROTERR	protocol error

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

Upon receipt of this indication, the application may only issue the following calls: s_synrq, s_uaborq, s_intrq, s_disrq or s_tkgrq (data token) to clear the error condition. All data is discarded and no sync point indications are given to the application until the error condition has been cleared.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-P-EXCEPTION-REPORT indication.

The Typed Data Functional Unit

Overview

The typed data functional unit supports the typed data transfer service.

The typed data functional unit comprises the following calls

s_typerq –	typed data request
s_typein –	receive typed data indication

s_typerq

NAME

s_typerq – typed data request

SYNOPSIS

```
int  s_typerq(sref,ptr,len,chain)
int  *sref;      (→)
char *ptr;       (→)
unsigned *len;  (→)
char *chain;     (→)
```

DESCRIPTION

’S_typerq’ asks for ’len’ bytes of typed user data from the area pointed to by ’ptr’ to be sent over the session connection with the local reference ’sref’. ’Chain’ specifies if this session interface data unit (SIDU) concludes a session service data unit (SSDU) or not, with either

S_MORE	This SIDU is not the end of an SSDU, or
S_END	This SIDU concludes an SSDU.

The SSDU is the unit of data exchanged between two session applications. The SIDU is the data unit exchanged at the local interface. The maximum length of an SIDU is implementation-dependent and can be queried using the s_info call.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

The ’chain’ parameter is useful for segmenting an SSDU that is too big to fit into one SIDU. Typed data is not subject to any token restrictions.

NOTE

If an application has sent an SIDU with the chain parameter set to S_MORE, no session request or response calls may be issued by the application, except s_uaborq, until the SSDU has been completed.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-TYPED-DATA request.

s_typein

NAME

s_typein – receive typed data indication

SYNOPSIS

```
int  s_typein(sref,ptr,len,chain)
int  *sref;     (→)
char *ptr;      (←)
unsigned *len;  (→)
char *chain;    (←)
```

DESCRIPTION

’S_typein’ receives typed user data announced via s_event for the session connection with the local reference ’sref’. ’Ptr’ points to an area of ’len’ bytes to which the typed data is written. If ’len’ is less than the length announced via s_event, the rest of the data must be received in one or a sequence of s_typein calls until all the announced data has been received, before further session calls can be issued. ’Chain’ points to a location in which the session service indicates if the received session interface data unit (SIDU) concludes a session service data unit (SSDU) or not, with either

S_MORE	This SIDU is not the end of an SSDU, or
S_END	This SIDU concludes an SSDU.

The SSDU is the unit of data exchanged between two session applications. The s_event always announces one SIDU, a data unit that is only meaningful at the local interface and has an implementation-dependent maximum size. If ’ptr’ is NULL, ’len’ bytes are discarded by the session service and not delivered to the application.

RETURN VALUES

>0	number of bytes still to be received in the announced SIDU
S_OK	one complete SIDU successfully received
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

Using the ’len’ parameter, an announced SIDU may be received in smaller segments with a sequence of s_typein calls.

NOTE

Even if the chain indicator is set to S_MORE, there is no minimum size of SIDU the user can be sure of receiving.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-TYPED-DATA indication.

The Capability Data Functional Unit

Overview

The capability data functional unit supports the capability data transfer service.

The capability data functional unit comprises the following calls

s_caprq	–	capability data request
s_capin	–	receive capability data indication
s_caprs	–	capability data response
s_capcf	–	receive capability data confirmation

s_caprq

NAME

s_caprq – capability data request

SYNOPSIS

```
int  s_caprq(sref,userdata)
int  *sref;                     (→)
struct s_udatas *userdata;      (→)
```

DESCRIPTION

’S_caprq’ requests capability data to be sent over the connection with the local reference ’sref’. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

The capability data request is answered by the partner application and the response arrives as a capability data confirmation announced by an s_event call. The s_caprq call is subject to the token restrictions in appendix A. The call can only be issued if the activity management functional unit was negotiated and no activity is in progress.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-CAPABILITY-DATA request.

s_capin

NAME

s_capin – receive capability data indication

SYNOPSIS

```
int  s_capin(sref,userdata)
int  *sref;                     (→)
struct s_udatas *userdata;      (←)
```

DESCRIPTION

’S_capin’ receives a capability data indication announced via s_event for the session connection with the local reference ’sref’. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;         * pointer to user data area    *
        unsigned  len;          * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

The capability data indication must be answered with a capability data response call (s_caprs).

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-CAPABILITY-DATA indication.

s_caprs

NAME

s_caprs – capability data response

SYNOPSIS

```
int  s_caprs(sref,userdata,chain)
int  *sref;                         (→)
struct s_udatas *userdata;          (→)
char chain;                         (→)
```

DESCRIPTION

’S_caprs’ answers the capability data indication received via s_capin for the session with the local reference ’sref’. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. ’Chain’ specifies if this function call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-CAPABILITY-DATA response.

s_capcf

NAME

s_capcf – receive capability data confirmation

SYNOPSIS

```
int  s_capcf(sref,userdata)
int  *sref;                     (→)
struct s_udatas *userdata;      (←)
```

DESCRIPTION

’S_capcf’ receives a capability data confirmation announced via s_event for the session with the local reference ’sref’, in response to a previously issued capability data request call. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;         * pointer to user data area    *
        unsigned  len;          * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-CAPABILITY-DATA confirm.

The Major Synchronize Functional Unit

Overview

The major synchronize functional unit supports the major synchronization point service.

The major synchronize functional unit comprises the following calls

s_majrq	–	sync major request
s_majin	–	sync major indication
s_majrs	–	sync major response
s_majcf	–	sync major confirmation

s_majrq

NAME

s_majrq – sync major request

SYNOPSIS

```
int  s_majrq(sref,syncp,userdata,chain)
int  *sref;                      (→)
long *syncp;                     (←)
struct s_udatas *userdata;       (→)
char chain;                      (→)
```

DESCRIPTION

'S_majrq' asks for a major synchronization point to be defined for the session with the local reference 'sref'. 'Syncp' points to a location to which the session service writes the identification number of the sync point. 'Userdata' is NULL if no user data is required, or points to an 's_udatas' struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

'Ptr' points to an area with 'len' bytes of user data to be transferred to the partner. If 'len' is 0, no user data is transferred. 'Chain' specifies if this function call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

If the activity management functional unit has been selected, this call may only be issued within an activity. The major sync point is defined when the sync major confirm (s_majcf) is received from the responding application. This event is announced by an s_event call. No further data may be requested until the s_majcf has been received. The s_majrq call is subject to the token restrictions in appendix A.

NOTE

It is up to the session user to ensure that the sync point number does not exceed 999998.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-SYNCH-MAJOR request.

s_majin

NAME

s_majin – receive sync major indication

SYNOPSIS

```
int  s_majin(sref,syncp,userdata)
int  *sref;                      (→)
long *syncp;                     (←)
struct s_udatas *userdata;       (←)
```

DESCRIPTION

’S_majin’ receives a request to define a major synchronization point announced via s_event for the session connection with the local reference ’sref’. ’Syncp’ points to a location to which the identification number of the sync point is written. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

The sync major indication must be answered with a sync major response call (s_majrs).

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-SYNCH-MAJOR indication.

s_majrs

NAME

s_majrs – sync major response

SYNOPSIS

```
int  s_majrs(sref,userdata,chain)
int  *sref;                        (→)
struct s_udatas *userdata;         (→)
char chain;                        (→)
```

DESCRIPTION

’S_majrs’ answers a sync major indication received via s_majin for the session with the local reference ’sref’. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. ’Chain’ specifies if this function call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-SYNCH-MAJOR response.

s_majcf

NAME

s_majcf – receive sync major confirm

SYNOPSIS

```
int  s_majcf(sref,userdata)
int  *sref;                       (→)
struct s_udatas *userdata;        (←)
```

DESCRIPTION

’S_majcf’ receives a sync major confirmation announced via s_event for the session connection with the local reference ’sref’, in response to a previously given sync major request. ’Userdata is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-SYNCH-MAJOR confirm.

The Resynchronize Functional Unit

Overview

The resynchronize functional unit supports the resynchronization service.

The resynchronize functional unit comprises the following calls

s_synrq	–	resync request
s_synin	–	resync indication
s_synrs	–	resync response
s_syncf	–	resync confirmation

s_synrq

NAME

s_synrq – resynchronize request

SYNOPSIS

```
int  s_synrq(sref,rtype,syncp,token,userdata)
int  *sref;                      (→)
char *rtype;                     (→)
long *syncp;                     (→)
char *token;                     (→)
struct s_udatas *userdata;       (→)
```

DESCRIPTION

’S_synrq’ requests an orderly reestablishment of communication for the session connection with the local reference ’sref’, e.g. after an error or if no response was sent by the partner application. ’Rtype’ specifies the type of resynchronization, with one of the following values:

S_ABANDON	synchronize to a defined state
S_RESTART	return to an agreed point; the sync point to be negotiated cannot be earlier than the last confirmed major sync point.
S_SET	synchronize to any specified valid sync point number

’Syncp’ is NULL (rtype = S_ABANDON) or points to a sync point number in the range 0-999999 (rtype = S_RESTART rtype = S_SET). ’Token’ points to the token assignment and its value is constructed by ORing values from the following list:

S_T_DATA	data token on responder side
S_T_MINOR	minor synchronize token on responder side
S_T_ACTIVITY	major activity token on responder side
S_T_RELEASE	release token on responder side
S_TC_DATA	data token on side chosen by responder
S_TC_MINOR	minor sync token on side chosen by responder
S_TC_ACTIVITY	major activity token on side chosen by responder
S_TC_RELEASE	release token on side chosen by responder

If a particular token has no value assigned to it, the token remains on the requester side or is not used in the current session. If all tokens in the session have no value, ’token’ may be NULL. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred.

This call may result in the loss of undelivered data.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

APPLICATION USAGE

The session is resynchronized when a resynchronize confirm (s_syncf) is received from the responding application. This event is announced by an s_event call. s_uaborq is the only call permissible before the s_syncf is received.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-RESYNCHRONIZE request.

s_synin

NAME

s_synin – receive resynchronize indication

SYNOPSIS

```
int  s_synin(sref,rtype,syncp,token,userdata)
int  *sref;                      (→)
char *rtype;                     (←)
long *syncp;                     (←)
char *token;                     (←)
struct s_udatas *userdata;       (←)
```

DESCRIPTION

’S_synin’ receives a request announced via s_event to resynchronize the session connection with the local reference ’sref’. ’Rtype’ points to a location to which the resynchronization type is written, as one of the following values:

S_ABANDON	synchronize to a defined state; the sync point number is greater than any previous value used in this session.
S_RESTART	return to an agreed point; the sync point to be negotiated cannot be earlier than the last confirmed major sync point.
S_SET	synchronize to any specified sync point number

’Syncp’ points to a location to which the identification number of the sync point is written. ’Token’ points to a location to which the token assignment is written. The value is constructed by ORing values from the following list:

S_T_DATA	data token on responder side
S_T_MINOR	minor synchronize token on responder side
S_T_ACTIVITY	major activity token on responder side
S_T_RELEASE	release token on responder side
S_TC_DATA	data token on side chosen by responder
S_TC_MINOR	minor sync token on side chosen by responder
S_TC_ACTIVITY	major activity token on side chosen by responder
S_TC_RELEASE	release token on side chosen by responder

If a particular token has no value assigned to it, the token remains on the requester side or is not used in the current session. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

The resynchronize indication must be answered with a resynchronize response call (s_synrs).

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-RESYNCHRONIZE indication.

s_synrs

NAME

s_synrs – resynchronize response

SYNOPSIS

```
int  s_synrs(sref,syncp,token,userdata,chain)
int  *sref;                      (→)
long *syncp;                     (→)
char *token;                     (→)
struct s_udatas *userdata;       (→)
char chain;                      (→)
```

DESCRIPTION

’S_synrs’ responds to a resynchronize request received via s_synrq for the session with the local reference ’sref’. ’Syncp’ points to the identification number of the sync point. ’Token’ points to the token assignment and its value is constructed by ORing values from the following list:

S_T_DATA	data token on responder side
S_T_MINOR	minor synchronize token on responder side
S_T_ACTIVITY	major activity token on responder side
S_T_RELEASE	release token on responder side

If a particular token has no value assigned to it, the token remains on the requester side or is not used in the current session. Only tokens where the requester has given the responder a choice may be specified. If all tokens in the session have no value, ’token’ may be NULL. ’Userdata’ is NULL if no user data is required, or points to an ’s_udatas’ struct with the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data          *
                };
```

’Ptr’ points to an area with ’len’ bytes of user data to be transferred to the partner. If ’len’ is 0, no user data is transferred. ’Chain’ specifies if this function call is to be concatenated with further session calls, with either

S_END	No further calls shall be concatenated, or
S_CONCAT	This call is immediately followed by a session call for the same session, which is to be concatenated with this call. (Rules for concatenation in appendix C.)

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.
S_STOP	call temporarily stopped due to flow control shortage; an S_GO event is announced via s_event once the call has been successfully completed and it is possible to continue with request and response calls for this session.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-RESYNCHRONIZE response.

s_syncf

NAME

s_syncf – receive resynchronize confirm

SYNOPSIS

```
int  s_syncf(sref,syncp,token,userdata)
int  *sref;                     (→)
long *syncp;                    (←)
char *token;                    (←)
struct s_udatas *userdata;      (←)
```

DESCRIPTION

’S_syncf’ receives a resynchronize confirmation announced via s_event for the session connection with the local reference ’sref’, in response to a previously issued resynchronize request. ’Synchp’ points to a location to which the identification number of the sync point is written. ’Token’ points to a location to which the token assignment is written. The token value is constructed by ORing values from the following list:

S_T_DATA	data token on responder side
S_T_MINOR	minor synchronize token on responder side
S_T_ACTIVITY	major activity token on responder side
S_T_RELEASE	release token on responder side

If a particular token has no value assigned to it, the token assignment was already specified by the requester or, if the responder was given the choice, the token assignment is on the requester side. ’Userdata’ is NULL or points to an ’s_udatas’ struct specifying the user data area and having the following layout:

```
struct s_udatas {
        char      *ptr;          * pointer to user data area    *
        unsigned  len;           * length of user data area     *
                };
```

’Ptr’ points to an area of ’len’ bytes to which the user data specified by the partner is written. If ’userdata’ is NULL or ’len’ is 0 or less than the length announced in the s_event call, all or the last part of the user data is ignored.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

RELATIONSHIP TO ISO 8326

Corresponds to the service primitive S-RESYNCHRONIZE confirm.

The Session Service Trace

Overview

The Session Service trace provides a means of recording all the proceedings at the interface and transport service level, including incoming and outgoing SPDUs. The trace may be switched on and off by an OSS application and is written to a trace file. It may then be read with the help of the trace evaluation program ’step’.

s_tron

NAME

s_tron — switch session trace on

SYNOPSIS

```
int  s_tron(name,tropt)
char *name;                          (→)
struct s_tropt1 *tropt;              (→)
```

DESCRIPTION

’S_tron’ switches on the internal session trace function. ’Name’ points to the trace file name selected by the user. An existing file with the same name is overwritten or extended, depending on the open mode. The information that can be traced includes session service calls, records defined by the session user, incoming and outgoing session protocol elements, some transport system calls and local internal calls. ’Tropt’ is NULL or points to a structure ’s_tropt1’ with the following layout:

```
struct s_tropt1 {
    char  s_trver;       * version of s_tropt layout        *
    char  s_trmode;      * open mode for trace file         *
    char  s_trsel;       * select traces to be switched on  *
    char  s_traopt;      * trace amount options             *
    long  s_mludata;     * max len for traced user data     *
    long  s_mldt;        * max len for traced data (DT SPDU)*
    long  s_mltd;        * max len for traced data (TD SPDU)*
              };
```

The version number ’s_trver’ is S_TROPT1.

The open mode ’s_trmode’ may be either

S_TR_NEW	create a new trace file, or
S_TR_EXT	extend old or create new file

The trace selection parameters ’s_trsel’, which can be combined, are as follows:

S_TR_USER	select the service user trace
S_TR_SERV	select the service trace
S_TR_PROT	select the protocol trace

In the trace amount options ’s_traopt’ the following may be specified:

S_TR_NOEV	trace s_event with NOEVENT result

’s_mludata’, ’s_mldt’ and ’s_mltd’ are the maximum lengths of user data, normal data or typed data to be traced, or S_TR_UNLIM if not limited.

If tropt is NULL, the default values are S_TR_NEW for ’s_trmode’, S_TR_USER+S_TR_PROT for ’s_trsel’, 0 for ’s_traopt’, S_TR_UNLIM for ’s_mludata’, and 0 for ’s_mldt’ and ’s_mltd’.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

The call will be unsuccessful if the trace function is already running. It is up to the user to ensure the uniqueness of the trace file names within the entire local system, e.g. by qualifying them with the process ID.

s_troff

NAME

s_troff – switch session trace off

SYNOPSIS

```
int    s_troff(NULL)
```

DESCRIPTION

’S_troff’ switches off the internal session trace function. If the trace function was not running, it is not regarded as an error. Following this call the trace file is closed and can be evaluated with the session trace evaluation program ’step’.

The parameter is reserved for future extensions.

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

If the trace function has not been switched off with this call when the user process terminates, the trace file will be closed but some trace records may be lost.

s_wutr

NAME

s_wutr – write user trace record

SYNOPSIS

```
int  s_wutr(sref,type,hdr,hdrlen,udata,udatalen)
int  *sref;                      (→)
int  type;                       (→)
char *hdr;                       (→)
int  hdrlen;                     (→)
char *udata;                     (→)
int  udatalen;                   (→)
```

DESCRIPTION

’S_wutr’ writes a user-defined trace record to the trace file opened by s_tron with trace selection parameter S_TR_USER. ’Sref’ points to a location containing the local reference of the session connection. If no session connection reference is to be used, ’sref’ points to S_NOSREF or may be NULL. ’Type’ specifies a trace record type in the range S_MINUTYPE to S_MAXUTYPE. The trace record to be written may consist of one or two parts, a header part and or a user data part. ’Hdr’ points to the header part with the length ’hdrlen’. ’Udata’ points to the user data part with the length ’udatalen’. Lengths may not exceed S_MAXUTRECL (12 Kbytes).

RETURN VALUES

S_OK	successful
S_ERROR	unsuccessful; a diagnostic code is available via the s_error call.

APPLICATION USAGE

The call will be unsuccessful if the trace function is not switched on. If the selection parameter S_TR_USER is not set, the call is ignored.

The session trace evaluation program STEP

NAME

step – session trace evaluation program

SYNOPSIS

The syntax for calling the session trace evaluation program is system-dependent (appendix D). ’Step’ accepts the following parameters:

```
[-h]  [-d]  [-l=nnn[k]]  [-s=n l m h]  [-cref=n]  [-ps=t s p a F]
[-f=hh[:mm[:ss]]]  [-t=hh[:mm[:ss]]]  [-m]  tracefile1 [tracefile2 ...]
```

meaning:

-h	outputs the command syntax to stdout.
-d	Data records are dumped; no analysis of session user data is performed.
-l=nnn	Dumps are limited to nnn bytes (rounded up to a multiple of 16); the length of the data is indicated in the message "output limit reached".
-l=nnnk	limitation of dumps to nnn Kbytes
-s=	security level for the analysis of session user data; default value is ’m’. n: no security level switched on l: Passwords are not listed. m: User identifications, account numbers and passwords are not listed. h: like ’m’, but file names are not listed either
-cref=n	Connection (session) reference, for which trace records should be evaluated; n = connection reference number
-ps=	Protocol layer, whose events (or PDUs) should be output; possible entries: t: transport events (without mass data transfer) s: session events i.e., with transport events and mass data transfer p: presentation events a: ACSE events F: FTAM events

	If the ps= option is not specified, all events are then output.
	It is recommended that trace records containing information on abnormal protocol operations, such as 'diagnostics' with FTAM, are always output. The evaluation routines of level 7 determine whether or not the records are output.
-f=hh:mm:ss	The time when the trace analysis begins:
	hh: hours mm: minutes ss: seconds
	If the option f is not set, the current time is taken; if either ss and m are not set, the the values ss=00 or mm=00 are taken.
	Default value: 00:00:00
-t=hh:mm:ss	The time where the trace analysis is ended; entries as for -f option.
	Default value: 23:59:59
-m	Chronological output of trace records from several simultaneously opened trace files, generated during multi-task operation.
	If the -m option is not set, the trace files specified are evaluated sequentially.
tracefile1 tracefile2 ...	trace file(s)

DESCRIPTION

The 'step' program evaluates a trace file containing a session service trace. The result of the evaluation is in printable form. 'Step' tries to evaluate the protocols (see -ps=) in the session user data unless the '-d'-option is set.

With all trace records the session references are available. The trace entries TCONRQ, TCONIN, TREDIN, TDISRQ and TDISIN were extended to include the transport reference. If several session connections are available for one transport connection, the first session reference (TCONRQ) and the last session reference (TDISRQ) differentiate themselves.

Diagnostic routine OSSD

NAME

ossd – OSS diagnostic routine

SYNOPSIS

’ossd’ accepts the following parameters:

```
⎧[-n]  filename  [[trmode]  [trsel]  [traopt]  [[mludata] [mldt] [mltd]]]⎫
⎨[-f]                                                                     ⎬
⎩[-i]                                                                     ⎭
```

Meaning of the options:

-n	generate trace options file
-f	delete trace option file
-i	display trace option file contents

Meaning of the parameters when using the -n option:

filename — Name of the first trace file to be created. This name is ignored, if the OSS application is called before ’ossd’ s_tron. In this case, the name given with s_tron remains valid.

trmode — Opening mode of the trace file; possible entries:

new:	create a new file or overwrite.
ext:	expand the existing file or create a new file.

Default value: new

trsel — Selects the traces which must be activated; possible entries:

user:	User trace
serv:	Service trace
prot:	Protocol trace

Several values can be connected with ’+’ in the order given above.

Default value: user+prot

traopt — Trace option; possible entries:

noev: s_event is also logged inclusive of S_NOEVENT

Default value: noev not set

mludata — Maximum amount of user data written to the trace file;

possible entries:

nnn:	a maximum of nnn bytes are written
unlim:	no restriction

mldt — Maximum amount of S-DATA data written to the trace file, possible values:

nnn:	a maximum of nnn bytes are written to the trace file
unlim:	no restriction

mltd — Maximum amount of S-TYPED-DATA data written to the trace file; possible values:

nnn:	a maximum of nnn bytes are written
unlim:	no restriction

DESCRIPTION

With ’ossd’, the OSS trace can be switched on independently of the OSS application. When ’ossd -n ...’ is called, ossd generates the trace option file SYOSS.TROPT in the current directory. The specified parameters are stored in this file. If this file is found in the current directory of the OSS application process when the first ’s_attach’ call is issued, OSS activates the trace with the parameters stored in the trace option file. The trace file generated is called ’filename.pid’ (pid = process number), if the OSS application had not already assigned a different name in an s_tron call.

APPLICATION USAGE

If all of the user data cannot be written in the trace file, the output is ended; at this point the message "trace limit reached" and the actual length of the user data are output.

Appendix A: Token Restrictions on Service Primitives

Service primitives	Data token	Sync minor token	Major activity token	Release token
S-RELEASE request	2	2	2	2
S-RELEASE response (negative)	nr	nr	nr	0
S-DATA request (half duplex)	1	nr	nr	nr
S-DATA request (duplex)	3	nr	nr	nr
S-CAPABILITY-DATA request	2	2	1	nr
S-TOKEN-GIVE request (data token)	1	nr	nr	nr
S-TOKEN-GIVE request (sync minor token)	nr	1	nr	nr
S-TOKEN-GIVE request (major act. token)	nr	nr	1	nr
S-TOKEN-GIVE request (release token)	nr	nr	nr	1
S-TOKEN-PLEASE request (data token)	0	nr	nr	nr
S-TOKEN-PLEASE request (sync minor token)	nr	0	nr	nr
S-TOKEN-PLEASE request (major act. token)	nr	nr	0	nr
S-TOKEN-PLEASE request (release token)	nr	nr	nr	0
S-CONTROL-GIVE request	2	2	1	2
S-SYNC-MINOR request	2	1	nr	nr
S-SYNC-MAJOR request	2	2	1	nr
S-U-EXCEPTION-REPORT request	0	nr	nr	nr
S-ACTIVITY-START request	2	2	1	nr
S-ACTIVITY-RESUME request	2	2	1	nr
S-ACTIVITY-INTERRUPT request	nr	nr	1	nr
S-ACTIVITY-DISCARD request	nr	nr	1	nr
S-ACTIVITY-END request	2	2	1	nr

Key:		
	0 :	Token available and not assigned to the SS-user who initiated the service primitive
	1 :	Token available and assigned to the SS-user who initiated the service primitive
	2 :	Token not available or token assigned to the SS-user who initiated the service primitive
	3 :	Token not available
	nr:	No restriction

Appendix B: Listing of the oss.h Include File

```
 * oss.h         OSS interface definitions                             *
 **********************************************************************
 **********************************************************************
 **********************************************************************
 **********************************************************************
 **********************************************************************
 *                                                                    *
 *                            O S S                                   *
 *                                                                    *
 **********************************************************************
 *                                                                    *
 *        O S S     I N T E R F A C E     D E F I N I T I O N S       *
 *                                                                    *
 **********************************************************************
 *                                                                    *
 *                  I N C L U D E     O S S . H                       *
 *                                                                    *
 **********************************************************************
 **********************************************************************
 **********************************************************************
 **********************************************************************
 **********************************************************************
 *
 (#) oss.h 3.07 92 07 22
*

#define S_OK              0       * function call successful         *
#define S_ERROR          -1       * function call unsuccessful,      *
                                  * due to permanent error           *
#define S_RETRY          -2       * function call unsuccessful,      *
                                  * due to temporary error           *
                                  * retry call later                 *
#define S_STOP           -3       * function call stopped due to     *
                                  * data flow control shortage       *
                                  * continue after event S_DATAGO    *

                                  * attach call value                *
#define S_NOTFIRST        2       * not first process of s-appl.     *

#define S_NOVALUE        -1L      * no value for sync points         *
#define S_NOUREF         -1       * no value for user references     *
#define S_NOSREF         -1       * no value for session reference   *

                                  * event call mode values           *
#define S_WAIT            0       * wait for next event to occur     *
#define S_CHECK           1       * check events                     *

                                  * connect result values            *
```

```
#define S_ACCEPT          0        * connect request accepted         *
#define S_REJECT          1        * connect request rejected         *
#define S_CONGEST         2        * connect request rejected due     *
                                   * to temporary congestion          *
#define S_PREJECT         3        * connect request rejected         *
                                   * from session service             *
#define S_PCONGEST        4        * connect request rejected due     *
                                   * to temporary congestion          *
#define S_PUNKNOWN        5        * connect request rejected due     *
                                   * to unknown application           *
#define S_PNATTACH        6        * connect request rejected due     *
                                   * to not attached application      *
#define S_PPVERS          7        * connect request rejected, since  *
                                   * protocol version not supported   *
#define S_PPICSREST       8        * connect request rejected due     *
                                   * to implementation restriction    *
                                   * stated in the PICS               *

                                   * release result values            *
#define S_AFFIRMATIVE     0        * request affirmed                 *
#define S_NEGATIVE        1        * negative release                 *
                                   * interface data unit values       *
#define S_END             0        * end of service data unit         *
#define S_MORE            1        * more data in this data unit      *
#define S_CONCAT          2        * calls are to be concatenated     *

                                   * sync point type values           *
#define S_EXPLICIT        0        * explicit confirm                 *
#define S_OPTIONAL        1        * optional confirm                 *

                                   * resync type values               *
#define S_RESTART         0        * restart return to last point     *
#define S_ABANDON         1        * abandon set new state            *
#define S_SET             2        * set to valid minor point         *

                                   * functional unit values           *
#define S_HDX        0x0001        * half duplex                      *
#define S_FDX        0x0002        * full duplex                      *
#define S_MINOR      0x0008        * minor synchronize                *
#define S_MAJOR      0x0010        * major synchronize                *
#define S_RESYNC     0x0020        * resynchronize                    *
#define S_ACTIVITY   0x0040        * activity management              *
#define S_NEGRELEASE 0x0080        * negotiated release               *
#define S_CAPABILITY 0x0100        * capability data                  *
#define S_EXCEPTIONS 0x0200        * exceptions                       *
#define S_TYPED      0x0400        * typed data as per                *
#define S_T62        0x1000        * fun. unit CCITT T.62             *
#define S_PVERS1     0x8000        * use session protocol version 1   *

                                   * token values                     *
#define S_T_DATA          1        * data token                       *
#define S_T_MINOR         4        * minor synchronize token          *
#define S_T_ACTIVITY     16        * major activity token             *
#define S_T_RELEASE      64        * release token                    *
#define S_T_ALL         (S_T_DATA | S_T_MINOR | S_T_ACTIVITY |\
                         S_T_RELEASE)
                                   *                                  *
#define S_TC_DATA         2        * data token choice                *
```

```
#define S_TC_MINOR        8        * minor synchronize choice         *
#define S_TC_ACTIVITY    32        * major activity token choice      *
#define S_TC_RELEASE    128        * release token choice             *
#define S_TC_ALL       (S_TC_DATA | S_TC_MINOR | S_TC_ACTIVITY |\
                        S_TC_RELEASE)

                                   * reason values                    *
#define S_TCDISCON        1        * transport disconnect             *
#define S_PROTERROR       4        * protocol error                   *
#define S_UNDEFINED       8        * undefined                        *
#define S_PICSREST       16        * restriction stated in the PICS   *
#define S_NOREASON        0        * non-specific error               *
#define S_OVERLOAD        1        * receiver ability jeopardized     *
#define S_SEQERR          3        * sequence error                   *
#define S_LOCALERR        5        * local SS-user error              *
#define S_PROCERR         6        * unrecoverable procedural error   *
#define S_DATATOKEN     128        * demand data token                *

                                   * list of possible events:         *
#define S_NOEVENT         0        * no session event occured         *
#define S_CONIN          13        * S-CONNECT indication             *
#define S_CONCF          14        * S-CONNECT confirm                *
#define S_RELIN           9        * S-RELEASE indication             *
#define S_RELCF          10        * S-RELEASE confirm                *
#define S_UABOIN        129        * S-U-ABORT indication             *
#define S_PABOIN        130        * S-P-ABORT indication             *
#define S_DATAIN          1        * S-DATA indication, announces     *
                                   * one interface data unit          *
#define S_TKGIN         131        * S-TOKEN-GIVE indication          *
#define S_TKPIN           2        * S-TOKEN-PLEASE indication        *
#define S_TYPEIN         33        * S-TYPED-DATA indication, an-     *
                                   * nounces one interface data unit  *
#define S_CAPIN          60        * S-CAPABILITY-DATA indication     *
#define S_CAPCF          61        * S-CAPABILITY-DATA confirm        *
#define S_MININ          49        * S-SYNCH-MINOR indication         *
#define S_MINCF          50        * S-SYNCH-MINOR confirm            *
#define S_MAJIN          41        * S-SYNCH-MAJOR indication         *
#define S_MAJCF          42        * S-SYNCH-MAJOR confirm            *
#define S_SYNIN          53        * S-RESYNCHRONIZE indication       *
#define S_SYNCF          34        * S-RESYNCHRONIZE confirm          *
#define S_STAIN          45        * S-ACTIVITY-START indication      *
#define S_RESIN          29        * S-ACTIVITY-RESUME indication     *
#define S_INTIN          25        * S-ACTIVITY-INTERRUPT indication  *
#define S_INTCF          26        * S-ACTIVITY-INTERRUPT confirm     *
#define S_DISIN          57        * S-ACTIVITY-DISCARD indication    *
#define S_DISCF          58        * S-ACTIVITY-DISCARD confirm       *
#define S_ENDIN         132        * S-ACTIVITY-END indication        *
#define S_ENDCF         133        * S-ACTIVITY-END confirm           *
#define S_CTGIN          21        * S-CONTROL-GIVE indication        *
#define S_UEXCIN         48        * S-U-EXCEPTION-REPORT indication  *
#define S_PEXCIN        134        * S-P-EXCEPTION-REPORT indication  *
#define S_GO            192        * S_DATA_GO indication             *
#define S_REDIN         193        * S-REDIRECT indication            *
#define S_TIMEINT       194        * time interrupt                   *

       struct s_udatas {
        char      *ptr;            * pointer to user data area        *
        unsigned  len;             * length of user data              *
```

```
                  };

       struct  s_cid   {           * layout of connection ID          *
        int     s_luref;           * length of SS-user reference      *
        char    s_uref[64];        * SS-user reference calling or     *
                                   * called                           *
        int     s_lcomref;         * length of common reference       *
        char    s_comref[64];      * common reference                 *
        int     s_laddref;         * length of additional ref         *
        char    s_addref[4];       * additional reference info        *
                };

       struct  s_ocid   {          * layout of connection ID          *
        int     s_lcguref;         * length of SS-user reference      *
        char    s_cguref[64];      * calling SS-user reference        *
        int     s_lcomref;         * length of common reference       *
        char    s_comref[64];      * common reference                 *
        int     s_laddref;         * length of additional ref         *
        char    s_addref[4];       * additional reference info        *
        int     s_lcduref;         * length of SS-user reference      *
        char    s_cduref[64];      * called SS-user reference         *
                };

       struct s_aid   {            * layout of activity ID            *
                int  s_lactid;     * length of identifier (max 6)     *
                char s_actid[6];   * activity identifier, trans-      *
                      };           * parent to session service        *

 **********************************************************************
 *                                                                    *
 *                      diagnostic codes                              *
 *                                                                    *
 **********************************************************************

 * non permanent errors:                                              *
#define S_NOMEM           1        * no memory available              *

 * invalid user call or protocol parameter:                           *
#define S_INVNAME       100        * invalid name length              *
#define S_INVEVMODE     101        * invalid event mode               *
#define S_INVSREF       102        * invalid session reference        *
#define S_INVCHAIN      103        * invalid chain parameter          *
#define S_INVCAT        104        * invalid concatenation            *
#define S_INVCID        105        * invalid connection ID            *
#define S_INVFUS        106        * invalid func. units parameter    *
#define S_INVTOKNI      107        * invalid token item               *
#define S_INVRSLT       108        * invalid result parameter         *
#define S_INVRSN        109        * invalid reason value             *
#define S_INVSYP        110        * invalid sync point parameter     *
#define S_INVSPT        111        * invalid sync point type          *
#define S_INVAID        112        * invalid activity identifier      *
#define S_INVMGLEN      113        * invalid message length           *
#define S_INVUDTA       114        * invalid user data parameter      *
#define S_ILLUDATA      115        * user data not permitted          *
#define S_INVQOS        116        * invalid quality of service param *
#define S_SYPOVFLW      117        * sync point overflow >= 999999    *
#define S_INVFRADDR     118        * invalid fromaddr                 *
#define S_INVTOADDR     119        * invalid toaddr                   *
```

```
#define S_PARNSUPP      120       * parameter not supported          *
#define S_INVPID        121       * redirect. to own or unknown proc.*
#define S_INVAREF       122       * invalid application reference    *
#define S_INVAUREF      123       * invalid appl. user reference     *
#define S_INVSUREF      124       * invalid session user reference   *
#define S_INVPVERS      125       * invalid session protocol version *
#define S_INVSSEL       126       * invalid session selector         *
#define S_INVTROPT      127       * invalid trace option parameter   *
#define S_INVUTYPE      128       * invalid user trace record type   *
#define S_INVUTRLEN     129       * invalid user trace record length *
 * invalid trace option in s_tron option structure                   *
#define S_INVTRVER      170       * invalid s_trver                  *
#define S_INVTRMODE     171       * invalid s_trmode                 *
#define S_INVTRSEL      172       * invalid s_trsel                  *
#define S_INVTRAOPT     173       * invalid s_traopt                 *
#define S_INVMLUDATA    174       * invalid s_trmludata              *
#define S_INVMLDT       175       * invalid s_trmldt                 *
#define S_INVMLTD       176       * invalid s_trmltd                 *

 * call sequence errors:                                             *
#define S_NOTSUPP       200       * function not supported           *
#define S_NOTATTACHED   201       * application not attached         *
#define S_OINCF         202       * outstanding or unexpected        *
                                  * ’in’ or ’cf’ call                *
#define S_STOPPED       203       * session in stopped state         *
#define S_MORESTATE     204       * session waits for more data      *
#define S_SPROTERR      205       * session protocol error           *
#define S_INVSTATE      206       * invalid state for this call      *
#define S_TRACEON       207       * trace already switched on        *
#define S_IVVER         208       * invalid OSS version number       *
#define S_TRNOTON       209       * trace not switched on            *

 * error in local environment:                                       *
#define S_SYSERR        300       * error on system call             *
                                  * error code returned in addinfo   *
#define S_TSERR         301       * error on transport system (TS)   *
                                  * call (addinfo contains more      *
                                  * information)                     *
#define S_CMXERR        S_TSERR   * supported for limited time period*
#define S_TSVER         302       * illegal TS version               *
#define S_CMXVER        S_TSVER   * supported for limited time period*
#define S_INVTIDULEN    303       * max TIDU length too short        *
#define S_ILLTS_USE     304       * user must not use TS and OSS     *
#define S_ILLCMXUSE     S_ILLTS_USE  * supp. for limited time period  *
#define S_NOLICENSE     305       * OSS license information missing  *
#define S_SHUTDOWN      306       * OSS shutdown indication          *

 * internal inconsistencies:                                         *
#define S_RLMERR        400       * release memory error             *
#define S_CCBQERR       402       * inconsistent ccb queue           *
#define S_INVPTIMEL     403       * prot.timer elapsed in inv. state *

 * error codes to be sent to remote session provider only:           *
#define S_ISPDULEN      500       * invalid SPDU length              *
#define S_INVSPDU       501       * invalid SPDU contents            *
#define S_MANDMISS      502       * mandatory parameter missing      *
#define S_INVTCDISC     503       * invalid transport disconnect     *
#define S_INVPOPT       504       * invalid protocol options         *
```

```
#define S_INVTSDU       505        * invalid maximum TSDU size        *
#define S_INVPV         506        * invalid protocol version         *
#define S_INVTKSI       507        * invalid token setting item       *
#define S_ILLRFLPR      508        * reflect parameter not permitted  *

 ********************************************************************
 *                                                                  *
 *                     trace option definitions                    *
 *                                                                  *
 ********************************************************************
 * value of s_trver parameter                                       *
#define S_TROPT1        1          * version of s_tropt1 layout       *
 * values of s_trmode parameter                                     *
#define S_TR_NEW        0          * create a new trace file          *
#define S_TR_EXT        1          * extend old or create new file    *
 * bit values of s_trsel parameter  (can be combined)               *
#define S_TR_USER       1          * switch on the service user trace *
#define S_TR_SERV       2          * switch on the service trace      *
#define S_TR_PROT       4          * switch on the protocol trace     *
 * bit values of s_traopt parameter  (can be combined)              *
#define S_TR_NOEV       1          * trace s_event with NOEVENT result*
 * value of s_trmludata, s_trmldt and s_trmltd to indicate no limit   *
#define S_TR_UNLIM      -1         * trace all data                   *
struct s_tropt1 {
        char  s_trver;             * version of s_tropt layout        *
        char  s_trmode;            * open mode for trace file         *
        char  s_trsel;             * select traces to be switched on  *
        char  s_traopt;            * trace amount options             *
        long  s_mludata;           * max len for traced userdata      *
        long  s_mldt;              * max len for traced data (DT SPDU)*
        long  s_mltd;              * max len for traced data (TD SPDU)*
              };
 ********************************************************************
 *                                                                  *
 *    definitions for writing user trace record function (s_wutr)     *
 *                                                                  *
 ********************************************************************
#define S_MINUTYPE      100        * minimum user trace record type   *
#define S_MAXUTYPE      2047       * maximum user trace record type   *
#define S_PRES_UTYPE    1000       * presentation user trace rec. type*
#define S_ACSE_UTYPE    1001       * ACSE user trace record type      *
#define S_FTAM_UTYPE    1002       * FTAM user trace record type      *
#define S_MAXURECL      (12*1024)  * maximum length of user trace rec.*
```

Appendix C: Rules for Concatenating Session Service Calls

Only the following call sequences may be concatenated:

```
s_starq  + s_datarq + s_minrq + s_tkgrq
s_resrq  + s_datarq + s_minrq + s_tkgrq
s_starq  + s_datarq + s_endrq
s_resrq  + s_datarq + s_endrq
s_starq  + s_datarq + s_majrq
s_resrq  + s_datarq + s_majrq
s_starq  + s_datarq + s_tkgrq
s_resrq  + s_datarq + s_tkgrq
s_starq  + s_minrq  + s_tkgrq
s_resrq  + s_minrq  + s_tkgrq
s_datarq + s_endrq  + s_tkgrq
s_datarq + s_endrs  + s_tkgrq
s_datarq + s_minrq  + s_tkgrq
s_datarq + s_minrs  + s_tkgrq
s_datarq + s_majrq  + s_tkgrq
s_datarq + s_majrs  + s_tkgrq
s_starq  + s_endrq
s_resrq  + s_endrq
s_starq  + s_majrq
s_resrq  + s_majrq
s_minrq  + s_tkgrq
s_minrs  + s_tkgrq
s_majrq  + s_tkgrq
s_majrs  + s_tkgrq
s_starq  + s_tkgrq
s_resrq  + s_tkgrq
s_endrq  + s_tkgrq
s_endrs  + s_tkgrq
s_datarq + s_tkgrq
s_minrs  + s_tkprq
s_majrs  + s_tkprq
s_intrs  + s_tkprq
s_disrs  + s_tkprq
s_synrs  + s_tkprq
s_caprs  + s_tkprq
s_uexcrq + s_tkprq
s_endrs  + s_tkprq
```

A concatenation (to potentially increase performance at the protocol level) is always possible at the interface. Whether a concatenation actually takes place at protocol level or not is transparent to the interface user and depends on the concatenating ability of the two session services involved.

Appendix D: BS2000-specific Characteristics

General

OSS V3.0 is a subsystem of BS2000 loadable with DSSM. It executes in the privileged processor state TPR and is called by TU application programs via an SVC interface or by TPR users via an BALR BASR interface.

OSS V3.0 sits on top of the CMX transport system interface.

Limit Values

The only limiting value defined by OSS V3.0 is the task-wide maximum number of applications and connections, 32000. No other limits exist for the number of applications or session connections except those imposed by the available memory space, BCAM or the operating system.

Installation

Installation Steps

- Reading in the following files:

SYSLNK.OSS.030:	contains the product OSS
SYSMSR.OSS.030:	MSG file (help texts)
SYSMSA.OSS.030:	MSG file (message texts)
SYSLIB.OSS.030:	contains the user include files for C, the user macros for ASSEMBLER, and the OSS runtime modules for C.

  The files SYSLIB.OSS.030, SYSMSR.OSS.030, SYSPRG.OSS.030.STEP, STEP, and SYSPRG.OSS.030.OSSD must be shareable.

- Adding the OSS subsystem declarations to the system's subsystem catalog:

  - Merging the file SYSSSD.OSS.030.BS2V9 (subsystem catalog entry for OSS) into the system’s subsystem catalog with UGEN for BS2000 V9.5.
  - Merging the file SYSSSD.OSS.030.BS2V10 (subsystem catalog entry for OSS) into the system’s subsystem catalog with UGEN for BS2000 V10.
  - Adding the file SYSSSC.OSS.BS2V11 (object definition for OSS) into the system’s subsystem catalog with SSCM for BS2000 V11.

- Reading in the REP file SYSREP.OSS.030. The REP file is accessed by DSSM when OSS is created and is accessed by OSS when it loads dynamically the BS2000-version dependent part of OSS.
- Reading in the NOREF file:
  - SYSLNK.OSS.030.NOREF for BS2000 V9.5
  - SYSNRF.OSS.030 from BS2000 V10 on.

  The NOREF file is needed for the REP file processing.

OSS expects the product library SYSLNK.OSS.030, the message files SYSMSR.OSS.030 and SYSMSA.OSS.030 and the REP file SYSREP.OSS.030 in the user-ID TSOS as the default option.

You can specify another user ID by the following modifications:

- Changing the file SYSSSD.OSS.030.BS2V95 resp. SYSSSD.OSS.030.BS2V10 (LIB, REP and MSG operands in the statement DSMATTR) in BS2000 V9.5 and V10.
- Changing the system’s subsystem catalog (parameter INSTALLATION-USERID in SSCM command Modify-SUBSYSTEM-ATTRIBUTES) in BS2000 V11.

The product library, message files, REP file, and NOREF file must be stored under the same user ID.

Subsystem Catalog Entry OSS(BS2000)

Catalog Entries for BS2000 V9 and BS2000 V10

Contents of the file SYSSSD.OSS.030.BS2V95

```
DSMATTR  OSS,VERSN=03.0,LIB= TSOS.SYSLNK.OSS.030
DSMATTR  OSS,VERSN=03.0,REP= TSOS.SYSREP.OSS.030,CHECK=STD
DSMATTR  OSS,VERSN=03.0,CREATIM=AFTSR,INIT=YOSINIT,STOPCOM=YOSSTCR
DSMATTR  OSS,VERSN=03.0,DEINIT=YOSDEIR,MSG= TSOS.SYSMSA.OSS.030
DSMCALL  OSS,VERSN=03.0,ENTRY=ENSVC189,MODE=SVC,VALUE=189,BYSYSEX=N,
             CONSCOP=PROG
DSMCALL  OSS,VERSN=03.0,ENTRY=YOSCALL,ACCESS=SYSTEM,CONSCOP=TASK
DSMCALL  OSS,VERSN=03.0,ENTRY=YOSSTRM,ACCESS=SYSTEM,CONSCOP=PROG
DSMAS    OSS,VERSN=03.0,CLASS=3 4,ACCESS=SYSTEM
DSMLINK  OSS,VERSN=03.0,LNKENT=YOSSG,REFSS=(CP)
DSMDEPND OSS,VERSN=03.0,DEPSS=(CP)
```

Contents of the file SYSSSD.OSS.030.BS2V10

```
DSMATTR  OSS,VERSN=03.0,LIB= TSOS.SYSLNK.OSS.030
DSMATTR  OSS,VERSN=03.0,REP= TSOS.SYSREP.OSS.030,CHECK=STD
DSMATTR  OSS,VERSN=03.0,CREATIM=AFTSR,INIT=YOSINIT,STOPCOM=YOSSTCR
DSMATTR  OSS,VERSN=03.0,DEINIT=YOSDEIR,MSG= TSOS.SYSMSA.OSS.030
DSMATTR  OSS,VERSN=03.0,INTVERS=YOSDINT,RESTART=YES
DSMCALL  OSS,VERSN=03.0,ENTRY=ENSVC189,MODE=SVC,VALUE=189,BYSYSEX=N,
             CONSCOP=PROG
DSMCALL  OSS,VERSN=03.0,ENTRY=YOSCALL,MODE=ISL,ACCESS=SYSTEM,CONSCOP=TASK
DSMCALL  OSS,VERSN=03.0,ENTRY=YOSSTRM,MODE=ISL,ACCESS=SYSTEM,CONSCOP=PROG
DSMAS    OSS,VERSN=03.0,CLASS=3 4,ACCESS=SYSTEM
DSMLINK  OSS,VERSN=03.0,LNKENT=YOSSG,REFSS=(CP)
DSMDEPND OSS,VERSN=03.0,DEPSS=(CP)
```

Notes:
The OSS and VERSN=03.0 parameters define the name and version number of the subsystem.

DSMATTR
This statement is used to define all the permanent characteristics of OSS that are required during installation and removal.

LIB= TSOS.SYSLNK.OSS.030
    Name of the module library from which DSSM loads OSS.

REP= TSOS.SYSREP.OSS.030
    Name of the REP file.

CHECK=STD
    At CREATE time, DSSM checks that the subsystem ID ’OSS 03.0’ is at the beginning of the advanced prelinked module YOSSG.

CREATIM=AFTSR

DSSM loads OSS automatically after "system ready".

INIT=YOSINIT
Name of the OSS-specific initialization routine.

STOPCOM=YOSSTCR
Name of the OSS-specific stop-commission routine.

DEINIT=YOSDEIR
Name of the OSS-specific deinitialization routine.

MSG= TSOS.SYSMSA.OSS.030
Name of the MSG file.

INTVERS=YOSDINT (BS2000 V10 specific parameter!)
Entry where the DSSM-interface is given.

RESTART=YES (BS2000 V10 specific parameter!)
If the holder task is terminated abnormally, the initialization routine must be called again.

DSMCALL
This statement is used to declare the OSS entry points.

Entry point ENSVC189:

ENTRY=ENSVC189	Name of SVC entry point
MODE=SVC	Branch via SVC
VALUE=189	SVC number
BYSYSEX=NO	The SVC cannot be called by system exit routines.
ACCESS=ALL	The SVC can be called by privileged and non-privileged processes (default value).
CONSCOP=PROG	DSSM disconnects the user at program end.

Entry point YOSCALL:

ENTRY=YOSCALL	Name of TPR entry point
MODE=ISL	The entry point is called via ISL (only for BS2000 V10)
ACCESS=SYSTEM	The entry point can only be used by privileged processes.
CONSCOP=TASK	DSSM disconnects the user at task end.

Entry point YOSTTRM:	Internal entry

DSMAS
This statement is used to make subsystem-specific address space declarations.

CLASS=3 4
OSS must be loaded in class 4 memory.

ACCESS=SYSTEM

OSS is to be loaded in privileged address space.

DSMLINK
This statement is used to control a subsystem linkage process.

LNKENT=YOSSG
Name of the object module to be loaded by DSSM

REFSS=(CP)
To resolve the external references, OSS only requires the BS2000 nucleus.

DSMDEPND
This statement is used to define all the subsystems required by OSS for execution.

DEPSS=(CP)
OSS requires the BS2000 nucleus for execution.

Catalog Entries for BS2000 V11

From BS2000 V11 the SSD object file SYSSSC.OSS.030.BS2V11 is delivered. Contents of SYSSSC.OSS.030.BS2V11 prepared by SSCM:

```
**************************************************************
* DSSM-SUBSYSTEM  NAME :      OSS     VERSION :    03.0      *
**************************************************************

GENREAL ATTRIBUTES :

 COPYRIGHT             :                            C.SIEMENS
 YEAR                  : 1992
 CREATION-TIME         : AFTER-SYSTEM-READY
 STOP-AT-SHUTDOWN      : YES
 SUBSYSTEM-HOLD        : ALLOWED
 STATE-CHANGE-COMMANDS : ALLOWED
 FORCED-STATE-CHANGE   : FORBIDDEN
 RESET                 : FORBIDDEN
 RESTART-REQUIRED      : YES
 VERSION-COEXISTENCE   : FORBIDDEN
 VERSION-EXCHANGE      : FORBIDDEN

INTERNAL ENTRIES :

 INIT ROUTINE      :  YOSINIT
 STOPCOM ROUTINE   :  YOSSTCR
 DEINIT ROUTINE    :  YOSDEIR
 DYNAMIC-CHECK     :    YOSSG
 INTERFACE-VERSION :  YOSDINT

MEMORY ATTRIBUTES :

 CLASS : SYSTEM-GLOBAL    SUBSYSTEM-ACCESS : SYSTEM

RELATED FILES :

 INSTALLATION USER-ID :    TSOS
 LIBRARY              : ..........................SYSLNK.OSS.030
 MESSAGE FILE         : ..........................SYSMSA.OSS.030
 SYNTAX FILE          : **** NOT SPECIFIED ****
 INFORMATION FILE     : **** NOT SPECIFIED ****
 REP FILE             : ..........................SYSREP.OSS.030
 REP FILE MANDATORY   : NO

LINK ATTRIBUTES :

 LINK ENTRY :    YOSSG
 AUTOLINK   : FORBIDDEN     UNRESOLVED : FORBIDDEN     CHECK REFERENCE : YES

REFERENCED DSSM-SUBSYSTEMS :

         NAME    LOWEST VERSION    HIGHEST VERSION
         ----    --------------    ---------------
            CP             00.0            99.9Z99

FUNCTIONAL DEPENDENCE WITH DSSM-SUBSYSTEMS :
```

```
         NAME    LOWEST VERSION    HIGHEST VERSION
         ----    --------------    ---------------
         **** NONE ****

SEPARATE-ADDRESS-SPACE FROM DSSM-SUBSYSTEMS :

         NAME            NAME            NAME            NAME
         ----            ----            ----            ----
         **** NONE ****

SHARED-HOLDER-TASK :        WITH DSSM-SUBSYSTEMS :

         NAME            NAME            NAME            NAME
         ----            ----            ----            ----
         **** NONE ****

SUBSYSTEM ENTRIES :

 NAME : ENSVC189
  MODE                 : SVC
  NUMBER               : 189
  CALL-BY-SYSTEM-EXIT  : FORBIDDEN
  FUNCTION NUMBER      : **** NOT SPECIFIED ****
  FUNCTION VERSION     : **** NOT SPECIFIED ****
  CONNECTION-ACCESS    : ALL
  CONNECTION-SCOPE     : PROGRAM
 NAME : YOSCALL
  MODE                 : ISL
  FUNCTION NUMBER      : **** NOT SPECIFIED ****
  FUNCTION VERSION     : **** NOT SPECIFIED ****
  CONNECTION-ACCESS    : SYSTEM
  CONNECTION-SCOPE     : TASK
 NAME : YOSSTRM
  MODE                 : ISL
  FUNCTION NUMBER      : **** NOT SPECIFIED ****
  FUNCTION VERSION     : **** NOT SPECIFIED ****
  CONNECTION-ACCESS    : SYSTEM
  CONNECTION-SCOPE     : PROGRAM
```

Notes:

CREATION-TIME : AFTER-SYSTEM-READY
DSSM loads OSS automatically after "system-ready"

STOP-AT-SHUTDOWN : YES
DSSM stops and unloads OSS automatically at system shutdown

SYSTEM-HOLD : ALLOWED
OSS can be held and deleted

STATE-CHANGE-COMMANDS : ALLOWED
The DSSM commands START-SUBSYSTEM, STOP-SUBSYSTEM, HOLD-SUBSYSTEM, RESUME-SUBSYSTEM for OSS can be issued from the console or by an userid with SUBSYSTEM-MANAGEMENT privilege.

FORCED-STATE-CHANGE : FORBIDDEN
STOP-SUBSYSTEM and HOLD-SUBSYSTEM with FORCED=YES are not allowed for OSS.

RESET : FORBIDDEN
START-SUBSYSTEM and RESUME-SUBSYSTEM with RESET=YES are refused for OSS.

RESTART-REQUIRED: YES
If the holder task is terminated abnormally, the initialization routine must be called again.

VERSION-COEXISTENCE : FORBIDDEN
Several versions of OSS must not be active in the same time.

VERSION-EXCHANGE : FORBIDDEN
Creation of OSS with the exchange-mode is not allowed.

The statements INTERNAL ENTRIES and RELATED FILES correspond to the statement DSMATTR in SYSSSD.OSS.030.BS2V95 resp. SYSSSD.OSS.030.BS2V10.

The statement MEMORY ATTRIBUTES corresponds to the statement DSMAS in SYSSSD.OSS.030.BS2V95 resp. SYSSSD.OSS.030.BS2V10.

The statements LINK-ATTRIBUTES and REFERENCED DSSM-SUBSYSTEMS correspond to the statement DSMLINK in SYSSSD.OSS.030.BS2V95 resp. SYSSSD.OSS.030.BS2V10.

The statement FUNCTIONAL DEPENDENCE WITH DSSM-SUBSYSTEMS corresponds to the statement DSMDEPND in SYSSSD.OSS.030.BS2V95 resp. SYSSSD.OSS.030.BS2V10.

Task Structure

Session applications and connections are managed by OSS on a task-specific basis.

Each session user task can open a number of session applications. More than one task can open the same session application.

Session connections are always assigned to a task. The redirect functions enable session connections (in certain circumstances only) to be passed from one task to another.

The only wait state an OSS user task can adopt is that initiated by an s_event() call with the parameter S_WAIT. This produces a wait state at a bourse if the bourse queue is empty.

Process Number Specifications (pid)

The process number that needs to be specified when reassigning session connections or in s_wake is the task sequence number or TSN, a 4-byte-long string in EBCDIC. The process number passed to OSS comprises these four characters as a 32-bit integer with the first character in the most significant position.

Example:
For a task with the TSN 1234 the process number 0xF1F2F3F4 should be specified.

Session Applications

Session applications with the same transport system application name can be opened by more than one task. For s_attach(), in the first task the return code S_OK is issued and in subsequent tasks S_NOTFIRST. Session connect requests (S_CONIN events) for this application are always issued to the task which has been opened the longest by the application. This means that, as long as it has not detached the application from OSS or terminated itself, the first task to open the application receives S_CONIN events, then the second, then the third etc.

Notes on the Application

In the rest condition OSS is in a wait state at a bourse. This has the following consequences for the OSS application:

- K2 and Cancel in this state take effect with a delay of max. 1 minute
- Timer contingencies or STXIT routines should not be used (owing to the delay). Instead, the OSS timer mechanism (s_timer) should be used.
- DCAM and OSS applications are not possible in the same task, owing to the absence of any synchronization of the waiting point mechanisms.
- When BS2000 interprocess communication (SOLSIG POSSIG) is used, it may be necessary to cancel an s_event wait state if it exists in the task to be initiated, with an s_wake call.

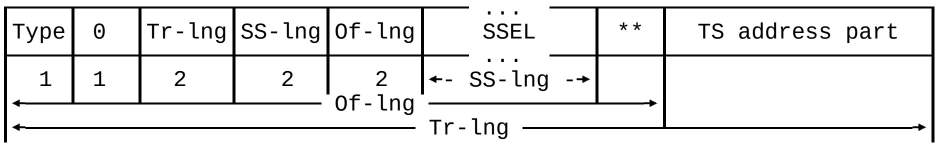
The following should also be noted:

- When the OSS subsystem is not loaded, the error code S_ERROR is returned for all session calls with s_error() = -1;
- If the OSS subsystem is unloaded, this is indicated by the return code S_ERROR for s_event with s_error() = S_SHUTDOWN. In this case all OSS applications must be closed also.

Address Structures in BS2000

Session application names (e.g. in ’s_attach’) and session addresses (e.g. in ’s_conrq’) must be passed in predefined structures, described in the include file cmx.h. These session address structures consist of a session selector part and transport address part. For alignment purposes there may be a gap between the session selector part and the transport system address part.

Layout of the session address structure



Type	T_SSAPINFO
Tr-lng	Transfer length (length of the session address structure)
SS-lng	Length of the session selector (0 to 16)
Of-lng	Offset length (8 to 24; must be a multiple of 4)
**	binary zeros for aligning the TS address structure (max. 3)

The structure of the TS address part is explained below. The structure for OSS V3.0 differs from the structures for OSS V1.0 V2.0. For OSS V1.0 V2.0 applications, the old structures can still be used, see manual "OSS (BS2000) V2.0".

Layout of the TS address structure for OSS V3.0

This structure is only allowed in OSS V3.0 applications. In OSS V1.0 V2.0 applications, the old structures must be used.

Address of a TS application (structure t_addr_nea)

Type	0	Length	Processor name	Application name	
1	1	2	8	8	Length in bytes

Type	T_MNMODE (= 1, for a local TS application)
Type	T_PAMODE (= 2, for a partner TS application)
Length	Length of the TS address structure (20 bytes)

In the case of Type =T_MNMODE, the field ’Processor name’ can be supplied with a virtual system (see manual "BS2000 XAF"). If the field is padded with blanks, the name of the local processor will be used.

Notes

- The 2-byte-long values in the header of the session address structure must be in the order high-order byte, low-order byte. The macros T_SETINFO and T_GETINFO in the include file cmx.h are available for this purpose.
- A session address without a session selector is defined with session selector length 0.
- Application and processor names less than 8 characters long must be written in the name field left-justified and padded up to 8 positions with blanks.

C Interface

The C interface for TU users comprises the 3 modules YOSSKN1, YOSSKN2 and YOSSWK, which need to be linked in to the session application programs. They are available in the library SYSLIB.OSS.030.

YOSSKN1 contains all the local functions except s_wake, as well as the functions of the kernel functional unit, half-duplex functional unit and negotiated release functional unit.

YOSSKN2 contains the functions of the typed data functional unit, capability data functional unit, minor, major and resynchronize functional units, the activity functional unit and exceptions functional unit.

YOSSWK contains the s_wake call.

Depending on the functions required by the session application, the module YOSSKN1, and if necessary YOSSKN2 and or YOSSWK, have to be linked in.
These three modules contain the code responsible for converting the C call interface into the SVC interface.

The user include file oss.h is available in the library SYSLIB.OSS.030.

ASSEMBLER Interface

Three macros for programming OSS applications in ASSEMBLER are available in the library SYSLIB.OSS.030. The macro YOSSEQU defines the necessary equates and must occur once, and once only, in a module. The YOSTROPT lets you define a DSECT for the trace options. The YOSS macro is used to call the OSS functions and define the associated parameter list. In order to use this interface you should be familiar with the interface description (in C syntax).

YOSSEQU macro

Name	Operation	Operands
———	YOSSEQU	———

The YOSSEQU macro has no operands. The equates are issued with the standard prefix YOS.

YOSTROPT macro

Name	Operation	Operands
Name	YOSTROPT	MF=(D,[xxx])

The prefix xxx (default YOS) may be 1 to 3 characters long.

YOSS macro

The YOSS macro supports the MF parameters MF = (L D C E).

Macro call in execute format

Name	Operation	Operands
[Name]	YOSS	[FUNC],MF=(E,addr (r))

FUNC	specifies the function, see table below. If this parameter is omitted, it is up to the user to enter the function number in the function list himself. The associated equates are defined in the YOSSEQU macro.
addr	symbolic address of the parameter list
r	register pointing to the parameter list.

The following table shows the values of FUNC, the equates for the associated function numbers and the corresponding C calls.

FUNC	Equate for function number	C call
ATTACH	YOSATTAC	s_attach
DETACH	YOSDETAC	s_detach
EVENT	YOSEVENT	s_event
INFO	YOSINFO	s_info
TIMER	YOSTIMER	s_timer
WAKE	YOSWAKE	s_wake
ERROR	YOSERROR	s_error
REDRQ	YOSREDRQ	s_redrq
REDIN	YOSREDIN	s_redin
TRON	YOSTRON	s_tron
TROFF	YOSTROFF	s_troff
WUTR	YOSWUTR	s_wutr
STOP	YOSSTP	s_stop
GO	YOSGO	s_go
CONRQ	YOSCONRQ	s_conrq
CONIN	YOSCONIN	s_conin
CONRS	YOSCONRS	s_conrs
CONCF	YOSCONCF	s_concf
RELRQ	YOSRELRQ	s_relrq
RELIN	YOSRELIN	s_relin
RELRS	YOSRELRS	s_relrs
RELCF	YOSRELCF	s_relcf
UABORQ	YOSUABOR	s_uaborq
UABOIN	YOSUABOI	s_uaboin
PABOIN	YOSPABOI	s_paboin
DATARQ	YOSDATAR	s_datarq
DATAIN	YOSDATAI	s_datain
TKGRQ	YOSTKGRQ	s_tkgrq
TKGIN	YOSTKGIN	s_tkgin
TKPRQ	YOSTKPRQ	s_tkprq
TKPIN	YOSTKPIN	s_tkpin
TYPERQ	YOSTYPER	s_typerq
TYPEIN	YOSTYPEI	s_typein
CAPRQ	YOSCAPRQ	s_caprq
CAPIN	YOSCAPIN	s_capin
CAPRS	YOSCAPRS	s_caprs
CAPCF	YOSCAPCF	s_capcf
MINRQ	YOSMINRQ	s_minrq
MININ	YOSMININ	s_minin
MINRS	YOSMINRS	s_minrs
MINCF	YOSMINCF	s_mincf
MAJRQ	YOSMAJRQ	s_majrq
MAJIN	YOSMAJIN	s_majin
MAJRS	YOSMAJRS	s_majrs
MAJCF	YOSMAJCF	s_majcf
SYNRQ	YOSSYNRQ	s_synrq
SYNIN	YOSSYNIN	s_synin
SYNRS	YOSSYNRS	s_synrs
SYNCF	YOSSYNCF	s_syncf
STARQ	YOSSTARQ	s_starq
STAIN	YOSSTAIN	s_stain
RESRQ	YOSRESRQ	s_resrq
RESIN	YOSRESIN	s_resin
INTRQ	YOSINTRQ	s_intrq
INTIN	YOSINTIN	s_intin
INTRS	YOSINTRS	s_intrs
INTCF	YOSINTCF	s_intcf
DISRQ	YOSDISRQ	s_disrq
DISIN	YOSDISIN	s_disin
DISRS	YOSDISRS	s_disrs
DISCF	YOSDISCF	s_discf
ENDRQ	YOSENDRQ	s_endrq
ENDIN	YOSENDIN	s_endin
ENDRS	YOSENDRS	s_endrs
ENDCF	YOSENDCF	s_endcf
CTGRQ	YOSCTGRQ	s_ctgrq
CTGIN	YOSCTGIN	s_ctgin
UEXCRQ	YOSUEXCR	s_uexcrq
UEXCIN	YOSUEXCI	s_uexcin
PEXCIN	YOSPEXCI	s_pexcin

Macro call in list format

Name	Operation	Operands
[Name]	YOSS	MF=L

This format has only one parameter, the MF parameter. Only the unit number, version number and the return code are entered in the standard header. All the other fields are prefilled with binary zeros.

Macro call in CSECT format

Name	Operation	Operands
[Name]	YOSS	MF=(C,[xxx])

The prefix xxx (default YOS) may be 1 to 3 characters long.

Macro call in DSECT format

Name	Operation	Operands
Name	YOSS	MF=(D,[xxx])

The name parameter must be specified in this call. The prefix xxx in the MF parameter (default YOS) may be 1 to 3 characters long.

Important programming notes

- Each parameter at the C interface corresponds to one field in the DSECT of the parameter list. It is up to the user to fill the fields required for the call. All other fields must be filled with binary zeros.
- The return information of an OSS call is in YOSRTINF. The associated equates are defined in the YOSSEQU macro.
- For the following parameters the user does not have to specify a pointer as in the C interface, but writes the value directly in the field, or the value is in the field after the OSS call:
  CHAIN, TOKEN, MTYPE, RTYPE, RESULT, AREF, AUREF, SUREF, SREF, FUNIT ADINF, DATAL, SYNCP, PID, REASON
- For the following parameters a pointer needs to be specified as well as the length:
  NAME, OPT, QOS, UAID, UCID, OCID, TOADDR, FRADDR
- For the USERDATA parameter you specify the pointer to the data, not the to the structure, and the length of the data:
- The values NOVALUE (for the SYNCP parameter), NOUREF (for the UREF, SUREF and AUREF parameters), NOSREF (for the SREF parameter) and TR_UNLIM (for the trace data lengths MLUDT, MLDT and MLTD) cannot be defined as ASSEMBLER equates (restriction imposed by the assembler). Instead, the constant F’-1’ should be used.

ASSEMBLER Macro Listings

YOSSEQU macro

```
         YOSSEQU
*
*        OSS HEADER VALUES
YOSUNIT  EQU   114                 OSS UNIT NUMBER
YOSVERS  EQU   1                   VERSION OF PARAMETER LIST LAYOUT
*
* FUNCTION NUMBER
YOSATTAC EQU   1                   S_ATTACH
YOSDETAC EQU   2                   S_DETACH
YOSEVENT EQU   3                   S_EVENT
YOSINFO  EQU   4                   S_INFO
YOSTIMER EQU   5                   S_TIMER
YOSWAKE  EQU   6                   S_WAKE
YOSERROR EQU   7                   S_ERROR
YOSREDRQ EQU   8                   S_REDRQ
YOSREDIN EQU   9                   S_REDIN
YOSTRON  EQU   10                  S_TRON
YOSTROFF EQU   11                  S_TROFF
YOSWUTR  EQU   12                  S_WUTR
YOSSTP   EQU   13                  S_STOP
YOSGO    EQU   14                  S_GO
YOSCONRQ EQU   30                  S_CONRQ
YOSCONIN EQU   31                  S_CONIN
YOSCONRS EQU   32                  S_CONRS
YOSCONCF EQU   33                  S_CONCF
YOSRELRQ EQU   34                  S_RELRQ
YOSRELIN EQU   35                  S_RELIN
YOSRELRS EQU   36                  S_RELRS
YOSRELCF EQU   37                  S_RELCF
YOSUABOR EQU   38                  S_UABORQ
YOSUABOI EQU   39                  S_UABOIN
YOSPABOI EQU   40                  S_PABOIN
YOSDATAR EQU   41                  S_DATARQ
YOSDATAI EQU   42                  S_DATAIN
YOSTKGRQ EQU   50                  S_TKGRQ
YOSTKGIN EQU   51                  S_TKGIN
YOSTKPRQ EQU   52                  S_TKPRQ
YOSTKPIN EQU   53                  S_TKPIN
YOSTYPER EQU   60                  S_TYPERQ
YOSTYPEI EQU   61                  S_TYPEIN
YOSCAPRQ EQU   70                  S_CAPRQ
YOSCAPIN EQU   71                  S_CAPIN
YOSCAPRS EQU   72                  S_CAPRS
YOSCAPCF EQU   73                  S_CAPCF
YOSMINRQ EQU   80                  S_MINRQ
YOSMININ EQU   81                  S_MININ
YOSMINRS EQU   82                  S_MINRS
YOSMINCF EQU   83                  S_MINCF
YOSMAJRQ EQU   90                  S_MAJRQ
YOSMAJIN EQU   91                  S_MAJIN
YOSMAJRS EQU   92                  S_MAJRS
YOSMAJCF EQU   93                  S_MAJCF
YOSSYNRQ EQU   100                 S_SYNRQ
```

```
YOSSYNIN EQU   101                S_SYNIN
YOSSYNRS EQU   102                S_SYNRS
YOSSYNCF EQU   103                S_SYNCF
YOSSTARQ EQU   110                S_STARQ
YOSSTAIN EQU   111                S_STAIN
YOSRESRQ EQU   112                S_RESRQ
YOSRESIN EQU   113                S_RESIN
YOSINTRQ EQU   114                S_INTRQ
YOSINTIN EQU   115                S_INTIN
YOSINTRS EQU   116                S_INTRS
YOSINTCF EQU   117                S_INTCF
YOSDISRQ EQU   118                S_DISRQ
YOSDISIN EQU   119                S_DISIN
YOSDISRS EQU   120                S_DISRS
YOSDISCF EQU   121                S_DISCF
YOSENDRQ EQU   122                S_ENDRQ
YOSENDIN EQU   123                S_ENDIN
YOSENDRS EQU   124                S_ENDRS
YOSENDCF EQU   125                S_ENDCF
YOSCTGRQ EQU   126                S_CTGRQ
YOSCTGIN EQU   127                S_CTGIN
YOSUEXCR EQU   130                S_UEXCRQ
YOSUEXCI EQU   131                S_UEXCIN
YOSPEXCI EQU   132                S_PEXCIN
*
*        OSS RETURN CODES
*
*        MAIN CODE VALUES
*
YOSOK    EQU   0                  S_OK       FUNCTION CALL SUCCESSFUL
YOSNFRST EQU   2                  S_NOTFIRST NOT FIRST PROCESS OF
*                                     SESSION APPLICATION (ATTACH)
YOSERR   EQU   255                S_ERROR    PERMANENT ERROR
YOSRETRY EQU   254                S_RETRY    TEMPORARY ERROR
YOSSTOP  EQU   253                S_STOP     FUNCTION CALL STOPPED DUE
*                                     TO DATA FLOW CONTROL SHORTAGE
*                                     CONTINUE AFTER EVENT S_DATAGO
YOSIVPL  EQU   256                INVALID PARAMETER LIST
*
*        SUBCODE1 DEFINES THE ERROR CLASS
*
YOSCOK   EQU   0                  NO ERROR
YOSCPERR EQU   1                  PERMANENT ERROR
YOSCSERR EQU   32                 SYSTEM ERROR
YOSCRERR EQU   64                 RECOVERABLE ERROR
YOSCSHRT EQU   128                RESOURCE SHORTAGE
*
*        VALUES OF CMODE PARAMETER
*
YOSWAIT  EQU   0                  WAIT FOR NEXT EVENT TO OCCUR
YOSCHECK EQU   1                  CHECK FOR EVENTS
*
*        VALUES OF RESULT PARAMETER
*
* CONNECT RESULT VALUES
YOSACC   EQU   0                  CONNECT REQUEST ACCEPTED
YOSREJ   EQU   1                  CONNECT REQUEST REJECTED
YOSCONG  EQU   2                  CONNECT REQUEST REJECTED DUE
```

```
*                                 TO TEMPORARY CONGESTION
YOSPREJ  EQU   3                  CONNECT REQUEST REJECTED
*                                 BY SESSION SERVICE
YOSPCONG EQU   4                  CONNECT REQUEST REJECTED DUE
*                                 TO TEMPORARY CONGESTION
YOSPUNKN EQU   5                  CONNECT REQUEST REJECTED
*                                 BECAUSE APPLICATION UNKNOWN
YOSNATT  EQU   6                  CONNECT REQUEST REJECTED
*                                 BECAUSE APPLICATION NOT ATTACHED
YOSPPVER EQU   7                  CONNECT REQUEST REJECTED BECAUSE
*                                 PROTOCOL VERSION IS NOT SUPPORTED
*
* RELEASE RESULT VALUES
YOSAFF   EQU   0                  REQUEST AFFIRMED
YOSNEG   EQU   1                  NEGATIVE RELEASE
*
*        VALUES OF CHAIN PARAMETER
*
YOSEND   EQU   0                  END OF SERVICE DATA UNIT
YOSMORE  EQU   1                  MORE DATA IN THIS DATA UNIT
YOSCAT   EQU   2                  CALLS ARE TO BE CONCATENATED
*
*        VALUES OF MTYPE PARAMETER
*
YOSEXPLC EQU   0                  EXPLICIT CONFIRM
YOSOPTC  EQU   1                  OPTIONAL CONFIRM
*
*        VALUES OF RTYPE PARAMETER
*
YOSRSTRT EQU   0                  RESTART; RETURN TO LAST POINT
YOSABAND EQU   1                  ABANDON; SET NEW STATE
YOSSET   EQU   2                  SET TO VALID MINOR POINT
*
*        VALUES OF FUNIT PARAMETER
*
YOSHDX   EQU   X’0001’            HALF DUPLEX
YOSFDX   EQU   X’0002’            FULL DUPLEX
YOSMIN   EQU   X’0008’            MINOR SYNCHRONIZE
YOSMAJ   EQU   X’0010’            MAJOR SYNCHRONIZE
YOSRESYN EQU   X’0020’            RESYNCHRONIZE
YOSACT   EQU   X’0040’            ACTIVITY MANAGEMENT
YOSNEGRL EQU   X’0080’            NEGOTIATED RELEASE
YOSCAP   EQU   X’0100’            CAPABILITY DATA
YOSEXCEP EQU   X’0200’            EXCEPTIONS
YOSTYPED EQU   X’0400’            TYPED DATA
YOST62   EQU   X’1000’            F U AS PER CCITT T.62
YOSPVER1 EQU   X’8000’            SESSION PROTOCOL V1
*
*        VALUES OF TOKEN PARAMETER
*
YOSTDAT  EQU   1                  DATA TOKEN
YOSTMIN  EQU   4                  MINOR SYNCHRONIZE TOKEN
YOSTACT  EQU   16                 MAJOR ACTIVITY TOKEN
YOSTREL  EQU   64                 RELEASE TOKEN
*
YOSTCDAT EQU   2                  DATA TOKEN CHOICE
YOSTCMIN EQU   8                  MINOR SYNCHRONIZE CHOICE
YOSTCACT EQU   32                 MAJOR ACTIVITY TOKEN CHOICE
```

```
YOSTCREL EQU   128                RELEASE TOKEN CHOICE
*
*        VALUES OF REASON PARAMETER
*
YOSRDISC EQU   1                  TRANSPORT DISCONNECT
YOSRPERR EQU   4                  PROTOCOL ERROR
YOSRUND  EQU   8                  UNDEFINED
YOSRNORS EQU   0                  NON-SPECIFIC ERROR
YOSROVL  EQU   1                  RECEIVER ABILITY JEOPARDIZED
YOSSEQER EQU   3                  SEQUENCE ERROR
YOSLOCER EQU   5                  LOCAL SS-USER ERROR
YOSPROER EQU   6                  UNRECOVERABLE PROCEDURAL ERROR
YOSDATTK EQU   128                DEMAND DATA TOKEN
*
*        VALUES OF TOADDR-LEN AND FRADDR-LEN PARAMETERS
*
YOSMYNML EQU   12                 LENGTH OF APPL’S OWN ADDRESS
YOSPADL  EQU   20                 LENGTH OF PARTNER’S ADDRESS
*
*        OSS EVENTS
*
YOSENOEV EQU   0                  NO SESSION EVENT OCCURRED
YOSECONI EQU   13                 S-CONNECT INDICATION
YOSECONC EQU   14                 S-CONNECT CONFIRMATION
YOSERELI EQU   9                  S-RELEASE INDICATION
YOSERELC EQU   10                 S-RELEASE CONFIRMATION
YOSEUABI EQU   129                S-U-ABORT INDICATION
YOSEPABI EQU   130                S-P-ABORT INDICATION
YOSEDATI EQU   1                  S-DATA INDICATION, ANNOUNCES
*                                 ONE INTERFACE DATA UNIT
YOSETKGI EQU   131                S-TOKEN-GIVE INDICATION
YOSETKPI EQU   2                  S-TOKEN-PLEASE INDICATION
YOSETYPI EQU   33                 S-TYPED-DATA INDICATION, ANNOUNCES
*                                 ONE INTERFACE DATA UNIT
YOSECAPI EQU   60                 S-CAPABILITY-DATA INDICATION
YOSECAPC EQU   61                 S-CAPABILITY-DATA CONFIRMATION
YOSEMINI EQU   49                 S-SYNCH-MINOR INDICATION
YOSEMINC EQU   50                 S-SYNCH-MINOR CONFIRMATION
YOSEMAJI EQU   41                 S-SYNCH-MAJOR INDICATION
YOSEMAJC EQU   42                 S-SYNCH-MAJOR CONFIRMATION
YOSESYNI EQU   53                 S-RESYNCHRONIZE INDICATION
YOSESYNC EQU   34                 S-RESYNCHRONIZE CONFIRMATION
YOSESTAI EQU   45                 S-ACTIVITY-START INDICATION
YOSERESI EQU   29                 S-ACTIVITY-RESUME INDICATION
YOSEINTI EQU   25                 S_ACTIVITY-INTERRUPT INDICATION
YOSEINTC EQU   26                 S-ACTIVITY-INTERRUPT CONFIRMATION
YOSEDISI EQU   57                 S-ACTIVITY-DISCARD INDICATION
YOSEDISC EQU   58                 S-ACTIVITY-DISCARD CONFIRMATION
YOSEENDI EQU   132                S-ACTIVITY-END INDICATION
YOSEENDC EQU   133                S-ACTIVITY-END CONFIRMATION
YOSECTGI EQU   21                 S-CONTROL-GIVE INDICATION
YOSEUEXI EQU   48                 S-U-EXCEPTION-REPORT INDICATION
YOSEPEXI EQU   134                S-P-EXCEPTION-REPORT INDICATION
YOSEGO   EQU   192                S-DATA-GO INDICATION
YOSEREDI EQU   193                S-REDIRECT INDICATION
YOSETINT EQU   194                TIME INTERRUPT
*
*        OSS DIAGNOSTIC CODES
```

```
*
*
* NON-PERMANENT ERRORS
YOSNOMEM EQU   1                 NO MEMORY AVAILABLE
*
* INVALID USER CALL OR PROTOCOL PARAMETER
YOSIVNAM EQU   100               INVALID NAME STRUCTURE
YOSIVEVM EQU   101               INVALID EVENT MODE
YOSIVSRF EQU   102               INVALID SESSION REFERENCE
YOSIVCHN EQU   103               INVALID CHAIN PARAMETER
YOSIVCAT EQU   104               INVALID CONCATENATION
YOSIVCID EQU   105               INVALID CONNECTION IDENTIFIER
YOSIVFUS EQU   106               INVALID FUNCTIONAL UNITS PARAMETER
YOSIVTKI EQU   107               INVALID TOKEN ITEMS
YOSIVRSL EQU   108               INVALID RESULT PARAMETER
YOSIVRSN EQU   109               INVALID REASON VALUE
YOSIVSYP EQU   110               INVALID SYNC POINT PARAMETER
YOSIVSPT EQU   111               INVALID SYNC POINT TYPE
YOSIVAID EQU   112               INVALID ACTIVITY IDENTIFIER
YOSIVMGL EQU   113               INVALID MESSAGE LENGTH
YOSIVUDT EQU   114               INVALID USER DATA PARAMETER
YOSILUDT EQU   115               USER DATA NOT PERMITTED
YOSIVQOS EQU   116               INVALID QUALITY OF SERVICE PARAM.
YOSSYPOV EQU   117               SYNC POINT OVERFLOW
YOSIVFAD EQU   118               INVALID FROM-ADDR
YOSIVTAD EQU   119               INVALID TO-ADDR
YOSPNSUP EQU   120               PARAMETER NOT SUPPORTED
YOSIVPID EQU   121               NO REDIRECTION TO OWN PROCESS
YOSIVARF EQU   122               INVALID APPLICATION REFERENCE
YOSIVAUR EQU   123               INVALID APPLICATION USER REFERENCE
YOSIVSUR EQU   124               INVALID SESSION USER REFERENCE
YOSIVPVE EQU   125               INVALID PROTOCOL VERSION
YOSIVSSL EQU   126               INVALID SESSION SELECTOR
YOSIVOPT EQU   127               INVALID TRACE OPTION PARAMETER
YOSIVUTP EQU   128               INVALID USER TRACE RECORD TYPE
YOSIVUTL EQU   129               INVALID USER TRACE RECORD LENGTH
*
* INVALID TRACE OPTION IN S_TRON OPTION STRUCTURE
YOSIVTVE EQU   170               INVALID TRACE VERSION
YOSIVTMD EQU   171               INVALID OPEN MODE FOR TRACE FILE
YOSIVTSL EQU   172               INVALID TRACE TO BE SWITCHED ON
YOSIVTOP EQU   173               INVALID TRACE AMOUNT OPTION
YOSIVTOD EQU   174               INVALID MAX. LENGTH FOR USER DATA
YOSIVTDT EQU   175               INVALID MAX. LENGTH FOR DATA
YOSIVTTD EQU   176               INVALID MAX. LENGTH FOR TYPED DATA
*
* CALL SEQUENCE ERRORS
YOSFNSUP EQU   200               FUNCTION NOT SUPPORTED
YOSANATT EQU   201               APPLICATION NOT ATTACHED
YOSOINCF EQU   202               OUTSTANDING IN OR CF CALL
YOSSSTOP EQU   203               SESSION IN STOPPED STATE
YOSSMORE EQU   204               SESSION AWAITING MORE DATA
YOSSPERR EQU   205               SESSION PROTOCOL ERROR
YOSIVSTA EQU   206               INVALID STATE FOR THIS CALL
YOSTRCON EQU   207               TRACE ALREADY SWITCHED ON
YOSIVVER EQU   208               INVALID OSS VERSION NUMBER
YOSTRNON EQU   209               TRACE IS NOT SWITCHED ON
*
```

```
* ERROR IN LOCAL ENVIRONMENT
YOSSYSER EQU   300               ERROR ON SYSTEM CALL
YOSTSER  EQU   301               ERROR ON TRANSPORT SYSTEM CALL
YOSTSVER EQU   302               ILLEGAL TRANSPORT SYSTEM VERSION
YOSIVTDL EQU   303               MAX. TIDU LENGTH TOO SHORT
YOSILTSU EQU   304               USER MUST NOT USE TRANSPORT SYSTEM
YOSNOLIC EQU   305               OSS LICENSE INFORMATION MISSING
YOSSHUTD EQU   306               OSS SHUTDOWN INDICATION
*
* INTERNAL INCONSISTENCIES
YOSRLMER EQU   400               RELEASE MEMORY ERROR
YOSCCBER EQU   402               INCONSISTENT CCB QUEUE
YOSIVPTM EQU   403               PROT. TIMER ELAPSED IN INV. STATE
*
* ERROR CODES SENT ONLY TO REMOTE SESSION PROVIDER
YOSIVSPL EQU   500               INVALID SPDU LENGTH
YOSIVSPD EQU   501               INVALID SPDU CONTENTS
YOSMPMIS EQU   502               MANDATORY PARAMETER MISSING
YOSIVTDS EQU   503               INVALID TRANSPORT DISCONNECT
YOSIVPRO EQU   504               INVALID PROTOCOL OPTIONS
YOSIVTSD EQU   505               INVALID MAXIMUM TSDU SIZE
YOSIVPRV EQU   506               INVALID PROTOCOL VERSION
YOSIVTSI EQU   507               INVALID TOKEN SETTING ITEM
YOSILRFP EQU   508               REFLECT PARAMETER NOT PERMITTED
*
*        TRACE OPTION DEFINITIONS
*
* VALUE OF TRVER PARAMETER
YOSTROP1 EQU   1                 VERSION OF DSECT YOSTROPT
*
* VALUES OF TRMODE PARAMETER
YOSTRNEW EQU   0                 CREATE A NEW TRACE FILE
YOSTREXT EQU   1                 EXTEND OLD OR CREATE NEW FILE
*
* BIT VALUES OF TRSEL PARAMETER (CAN BE COMBINED)
YOSTRUSR EQU   1                 SWITCH SERVICE USER TRACE ON
YOSTRSRV EQU   2                 SWITCH SERVICE TRACE ON
YOSTRPRT EQU   4                 SWITCH PROTOCOL TRACE ON
*
* BIT VALUE OF TRAOPT PARAMETER
YOSTRNEV EQU   1                 TRACE S_EVENT WITH NO-EVENT RESULT
*
*        DEFINITIONS FOR FUNCTION S_WUTR
*
YOSTMIUS EQU   100               MINIMUM USER TRACE RECORD TYPE
YOSTMXUS EQU   2047              MAXIMUM USER TRACE RECORD TYPE
YOSTPS   EQU   1000              PRESENTATION TRACE RECORD TYPE
YOSTACSE EQU   1001              ACSE TRACE RECORD TYPE
YOSTFTAM EQU   1002              FTAM TRACE RECORD TYPE
*
YOSMXUTL EQU   12288             MAX. LENGTH OF USER TRACE RECORDS
*
               *,YOSSEQU  308    920901
```

YOSTROPT macro

```
TROPT    YOSTROPT MF=(D,OSS)
TROPT    DSECT
OSSTROPT DS    0A
OSSTRVER DS    X              VERSION OF DSECT
OSSTRMOD DS    X              OPEN MODE FOR TRACE FILE
OSSTRSEL DS    X              SELECT TRACES TO BE SWITCHED ON
OSSTRAOP DS    X              TRACE AMOUNT OPTIONS
OSSMLUDT DS    F              MAX. LENGTH FOR TRACED USER DATA
OSSMLDT  DS    F              MAX. LENGTH FOR TRACED DATA (DT SPDU)
OSSMLTD  DS    F              MAX. LENGTH FOR TRACED DATA (TD SPDU)
OSSTRRES DS    16X            RESERVED FOR FUTURE EXTENSIONS
OSSTROPL EQU   *-OSSTROPT     LENGTH OF DSECT
               *,YOSTROPT   300    920318
```

List format of the YOSS macro

```
YOSLIST  YOSS  MF=L
YOSLIST  DS    0F
         FHDR  MF=L,UNIT=YOSUNIT,FUNCT=0,VERS=YOSVERS
               *,FHDR VERSION 002  26. 1. 1987
         DS    0A
         DS    0XL8                GENERAL OPERAND LIST HEADER
         DC    AL2(YOSUNIT)        FUNCTION UNIT NUMBER
         DC    AL1(0)         FUNCTION NUMBER
         DC    AL1(YOSVERS)        FUNCTION INTERFACE VERSION NUMBER
         DC    X’FFFFFFFF’         RETURN CODE NOT VALID
         DS    0F
         DC    X’0’        CHAIN
         DC    X’0’        MTYPE RTYPE RESULT
         DC    X’0’        TOKEN
         DC    X’0’        RESERVED
         DC    F’0’        RTINF
         DC    F’0’        AREF
         DC    F’0’        UREF AUREF SUREF
         DC    F’0’        SREF
         DC    F’0’        TYPE CMODE FUNITS
         DC    F’0’        ADINF UDATAL SYNCP TIME PID REASON MAXSIDU
         DC    A(0)        USERDATA-PTR PTR NAME-PTR
         DC    F’0’        USERDATA-LEN LEN NAME-LEN
         DC    A(0)        HEADER-PTR OPTIONS-PTR QOS-PTR UACTID-PTR
         DC    F’0’        HEADER-LEN OPTIONS-LEN QOS-LEN UACTID-LEN
         DC    A(0)        UCID-PTR OLDCID-PTR
         DC    F’0’        UCID-LEN OLDCID-LEN
         DC    A(0)        TOADDR-PTR OLDACTID-PTR
         DC    F’0’        TOADDR-LEN OLDACTID-LEN
         DC    A(0)        FRADDR-PTR
         DC    F’0’        FRADDR-LEN
               *,YOSS       310    921020
```

CSECT format of the YOSS macro

```
YOSPAR   YOSS  MF=C
YOSHDR   FHDR  MF=(C,YOS)
               *,FHDR VERSION 002  26. 1. 1987
YOSHDR   DS    0A
YOSFHE   DS    0XL8          0   GENERAL OPERAND LIST HEADER
*
YOSIFID  DS    0A            0   INTERFACE IDENTIFIER
YOSFCTU  DS    AL2           0   FUNCTION UNIT NUMBER
*                                BIT 15    HEADER FLAG BIT
*                                MUST BE RESET UNTIL FURTHER NOTICE
*                                BITS 14-12 NOT USED MUST BE RESET
*                                BITS 11-0  REAL FUNCTION UNIT NUMBER
YOSFCT   DS    AL1           2   FUNCTION NUMBER
YOSFCTV  DS    AL1           3   FUNCTION INTERFACE VERSION NUMBER
*
YOSRET   DS    0A            4   GENERAL RETURN CODE
*
* GENERAL_RETURN_CODE CLEARED (X’00000000’) MEANS
* REQUEST SUCCESSFULLY PROCESSED AND NO ADDITIONAL INFORMATION
*
YOSSRET  DS    0AL2          4   SUB RETURN CODE
YOSSR2   DS    AL1           4   SUB RETURN CODE 2
* ALWAYS CLEARED (X’00’) IF MAIN_RETURN_CODE IS X’FFFF’
YOSSR1   DS    AL1           5   SUB RETURN CODE 1
*
* SPECIAL LAYOUT OF LINKAGE_SUB_RETURN_CODE_1 (XX IN X’00XXYYYY’)
*
*       ALL EQUATES LESS  X’40’  ———> NO RETRY POSSIBLE
*
YOSRFNS  EQU   X’01’             CALLED FUNCTION NOT SUPPORTED
YOSRFNA  EQU   X’02’             CALLED FUNCTION NOT AVAILABLE
YOSRVNA  EQU   X’03’             INTERFACE VERSION NOT SUPPORTED
YOSRAER  EQU   X’04’             ALIGNMENT ERROR
*
*       ALL EQUATES GREATER OR EQUAL X’40’ AND LESS THAN X’80’
*       ———> CORRECT AND RETRY
*
YOSRCAR  EQU   X’40’             CORRECT AND RETRY
YOSRECR  EQU   X’41’             SUBSYSTEM (SS) MUST BE CREATED
*                                EXPLICITLY BY CREATE-SS
YOSRECN  EQU   X’42’             SS MUST BE EXPLICITLY CONNECTED
*
*       ALL EQUATES GREATER OR EQUAL X’80’ ———> WAIT AND RETRY
*
YOSRWAR  EQU   X’80’             WAIT AND RETRY
YOSRTNA  EQU   X’81’             SS TEMPORARY NOT AVAILABLE
YOSRDH   EQU   X’82’             SS IN DELETE   HOLD
*
YOSMRET  DS    AL2           6   MAIN RETURN CODE
*
* SPECIAL LAYOUT OF LINKAGE_MAIN_RETURN_CODE (YYYY IN X’00XXYYYY’)
*
YOSRLNK  EQU   X’FFFF’           LINKAGE ERROR   REQ. NOT PROCESSED
YOSFHL   EQU   8             8   GENERAL OPERAND LIST HEADER LENGTH
*
         DS    0F
```

```
YOSCHAIN DS    X           CHAIN
YOSMTYPE DS    0X          MTYPE      SYNC POINT TYPE
YOSRTYPE DS    0X          RTYPE      RESYNC POINT TYPE
YOSRESLT DS    X           RESULT
YOSTOKEN DS    X           TOKEN
         DS    X           RESERVED
YOSRTINF DS    F           RETURN INFORMATION
YOSAREF  DS    F           AREF       APPLICATION REFERENCE
YOSUREF  DS    0F          UREF       USER REFERENCE
YOSAUREF DS    0F          AUREF      APPLICATION USER REFERENCE
YOSSUREF DS    F           SUREF      SESSION USER REFERENCE
YOSSREF  DS    F           SREF       SESSION REFERENCE
YOSTYPE  DS    0F          TYPE       TRACE RECORD TYPE
YOSCMODE DS    0F          CMODE      EVENT CALL MODE
YOSFUNIT DS    F           FUNITS     FUNCTIONAL UNITS
YOSADINF DS    0F          ADDITIONAL DIAGNOSTIC INFORMATION
YOSDATAL DS    0F          UDATAL     EVENT USER DATA LENGTH
YOSSYNCP DS    0F          SYNCP      SYNC POINT
YOSTIME  DS    0F          TIME
YOSPID   DS    0F          PID        PROCESS ID
YOSREASN DS    0F          REASON
YOSMSIDU DS    F           MAXSIDU
YOSUDTA  DS    0A          USERDATA-PTR
YOSPTR   DS    0A          PTR
YOSNAME  DS    A           NAME-PTR
YOSUDTAL DS    0F          USERDATA-LEN
YOSLEN   DS    0F          LEN
YOSNAMEL DS    F           NAME-LEN
YOSHEADR DS    0A          HEADER-PTR
YOSOPT   DS    0A          OPTIONS-PTR
YOSQOS   DS    0A          QOS-PTR
YOSUAID  DS    A           UACTID-PTR
YOSHDRL  DS    0F          HEADER-LEN
YOSOPTL  DS    0F          OPTIONS-LEN
YOSQOSL  DS    0F          QOS-LEN
YOSUAIDL DS    F           UACTID-LEN
YOSUCID  DS    0A          UCID-PTR
YOSOCID  DS    A           OLDCID-PTR
YOSUCIDL DS    0F          UCID-LEN
YOSOCIDL DS    F           OLDCID-LEN
YOSTOAD  DS    0A          TOADDR-PTR
YOSOAID  DS    A           OLDACTID-PTR
YOSTOADL DS    0F          TOADDR-LEN
YOSOAIDL DS    F           OLDACTID-LEN
YOSFRAD  DS    A           FRADDR-PTR
YOSFRADL DS    F           FRADDR-LEN
YOSPARL  EQU   *-YOSHDR    LENGTH OF PARAMETER LIST
         ORG   YOSHDR
YOSPAR   DS    0XL(YOSPARL)
         ORG   YOSHDR+YOSPARL
               *,YOSS     310    921020
```

Diagnostics with s_error

The s_error call supplies the cause of an error. The return values are described in the oss.h include file. The additional information for S_TSERR is described in the cmx.h include file. In the event of S_SYSERR, the additional information returned has the following layout:

CI	Error code

CI	Call info byte
1	DMS call; "Error code" is the DMS error code
255	internal system error; "Error code" is X’000000’

OSS Messages

OSS issues the following messages at the console:

`OSS0254 WRONG SUBSYSTEM DECLARATIONS`

Meaning
: The OSS subsystem catalog declarations are not compatible to the BS2000 version.

Action
: Create new subsystem catalog with valid declarations of OSS.

`OSS0255 SYSTEM ERROR. ERROR CODE (&00)`

Meaning
: A system error has occurred.

Action
: Compile diagnostic documents and notify system diagnostics staff.

Session Trace Analysis in BS2000

The session trace analysis in BS2000 is performed by a procedure with the default name STEP. This calls the trace analysis program proper OSS.STEP.

```
                                                   ⎡’(SYSLST)’⎤
DO STEP,trace-filename[,OPTS=’parameter’][,OUTPUT=⎨’(SYSOUT)’ ⎬]
                                                   ⎣filename  ⎦
```

trace-filename	is the name of the trace file to be analyzed.
OPTS=	specifies the analysis parameters. The possible values and their meaning are described on page 120. Default: no parameters.
OUTPUT=	specifies the output medium for the edited trace.
’(SYSLST)’	Output to SYSLST (default)
’(SYSOUT)’	Output to SYSOUT
filename	Output to the specified file

It is also possible to specify more than one trace file, see the following example:

```
DO STEP,’tracefile1 -
tracefile2 -
tracefile3’,OUTPUT=outputfile
```

Appendix E: Sample Program for a Simple Session Run

The following program shows how a session application can control its connections and how the requisite parameters are passed to OSS functions or obtained from OSS. The program has the following structure:

Header:

An application attaches itself to the session service. The session service issues a fixed number (SCONN) of connection requests to one or more different partners in a ’for’ loop. A timer is activated to prevent the application process waiting for ever for session events. The timer is also responsible for monitoring when all the connections have been cleared down again.

Session run:

The connections are controlled by incoming events in a ’while’ loop. After the s_event call, the connection control block associated with the supplied sref is identified - where possible. The control blocks contain static memory for the session parameters and, in addition, some information concerning the current status of the session. This information comprises the ’state’ as per ISO Service Definition 8326, the next action to be taken and the sending ability of the connection.

The program initiates the next action in the relevant session in accordance with the event announced.

During the run, the application initiates an activity on each connection, alternately sends data and minor sync requests 100 times and then terminates the activity. Incoming minor sync point confirmations are accepted, but not necessary. Finally the application clears down all connections.

Trailer:

As soon as no session events occur between two time monitoring intervals, the program leaves the while loop and the application is detached from the session service. This implicitly destroys any connections that may still exist due to incorrect execution.

For the program to execute successfully, there must be a ’passive’ application on the partner’s side to receive all requests and data, and answer events ’requiring confirmation’ with an appropriate response.

```
 * necessary include files :          *

#include <stdio.h>
#include <cmx.h>
#include <oss.h>

 * definitions :                      *

#define     NULL       0
#define     FALSE      0
#define     TRUE

#define     SCONN     10               * number of connections      *

                                       * states in accordance with  *
                                       *  ISO Service Def. 8326     *
#define     STA01      1               * idle; no connection        *
#define     STA02A     2               * wait for S_CONCF           *
#define     STA03      3               * wait for S_RELCF           *
#define     STA04B     4               * wait for S_ENDCF           *
#define     STA713     5               * data transfer state        *

                                       * actions to be executed     *
#define     SENDDATA   1               * send data                  *
#define     SETSYP     2               * set a minor sync point     *
#define     ENDACT     3               * request end of activity    *

union    t_address sapplic;            * prog.’s own application    *
int      aref;                         * application reference      *
int      sref;                         * announced session ref.     *
int      uref;                         * announced user reference   *
unsigned udatal;                       * announced data length      *
unsigned sec = 600;                    * limit for timeout          *
int      errcode;                      * error code                 *
int      addinfo;                      * additional error code      *
char     timeout  = FALSE;             * a time interrupt occurred  *
char     no_event = FALSE;             * no event occurred in the   *
                                       *  last time interval        *
struct sctr {                          * session control struct     *
       union    t_address toaddr;      * session parameters         *
       int      sref;
       struct   s_cid ucid;
       int      funits;
       long     syncp;
       char     token;
       struct   s_udatas userdata;
       char     result;
       char     chain;
       char     mtype;
       struct   s_aid uactid;
                                       * session environment        *
       int      state;
       char     next_action;           * next action to be executed *
       char     stopped; } sc[SCONN];
struct sctr* scp;
int      i;                            * index variable             *
int      rc;                           * return code                *
int      ewa;                          * event watcher              *
char     usdata[512];                  * field for user data        *

char     comref[]="Example of a common reference";
char     regend[]="This is a regular end of session";

 * processing part :                  *

main()

{
 * storing an application name in the field 'sapplic' in accordance *
 * with the rules of the underlying transport system ...            *

rc = s_attach(&aref, NULL, &sapplic, NULL);
if (rc == S_ERROR) ...                 * error handling             *

for (i = 0; i < SCONN; ++i)            * connection requests        *
  {
  scp = &sc[i];
 * storing the partner address in the field 'scp->toaddr' in accor- *
 * dance with the rules of the underlying transport system ...      *

  scp->ucid.s_luref =                  * s_addref,s_uref-field      *
    scp->ucid.s_laddref = 0;           * not used                   *

  strcpy(scp->ucid.s_comref,comref);
  scp->ucid.s_lcomref = strlen(comref);
  scp->funits = S_HDX+S_MINOR+S_ACTIVITY+S_EXCEPTIONS;
  scp->token  = 0;
  scp->userdata.len = 0;

 * connection request      *
  rc = s_conrq(&scp->sref, &i, &aref, &scp->toaddr, &scp->ucid,
                &scp->funits, NULL, NULL, &scp->token, &scp->userdata);
  if (rc == S_OK) scp->state = STA02A;
  else            scp->state = STA01;
  scp->stopped = FALSE;
  scp->next_action = SENDDATA;
  }

rc = s_timer(sec);                     * set timer so system does   *
                                       * not wait for ever          *
while (no_event == FALSE)              * an event occurred during   *
  {                                    * the last 2 time intervals  *
  uref = S_NOUREF;
  ewa = s_event(&sref, &uref, S_WAIT, &udatal);
  if (ewa != S_TIMEINT)
    timeout = FALSE;
  if (uref != S_NOUREF)
  {
    scp = &sc[uref];
    scp->sref = sref;
  scp->userdata.ptr = usdata;      * prepare user data struct   *
  scp->userdata.len = udatal;
}
else scp = NULL;

switch(ewa)
  {
  case S_NOEVENT:
              break;
  case S_CONCF: rc = s_concf(&scp->sref, &scp->toaddr, &scp->ucid,
                          &scp->result,&scp->funits,NULL,
                          &scp->syncp,&scp->token,&scp->userdata);
              if (rc == S_ERROR) ...       * error handling     *
              scp->state = STA713;

              strcpy(scp->uactid.s_actid,"ACT 1");
              scp->uactid.s_lactid = 5;
              rc = s_starq(&scp->sref, &scp->uactid, NULL, S_END);
              if (rc == S_ERROR) ...       * error handling     *
              if (rc == S_OK)
                send();
              else scp->stopped = TRUE;
              break;
  case S_MINCF: rc = s_mincf(&scp->sref,&scp->syncp,&scp->userdata);
              if (rc == S_ERROR) ...       * error handling     *
              if (scp->stopped == FALSE)
                send();
              break;
  case S_ENDCF: rc = s_endcf(&scp->sref,&scp->userdata);
              if (rc == S_ERROR) ...       * error handling     *
              scp->userdata.ptr = regend;
              scp->userdata.len = strlen(regend);
              rc = s_relrq(&scp->sref,&scp->userdata);
              if (rc == S_ERROR) ...       * error handling     *
              scp->state = STA03;
              if (rc == S_STOP)
                scp->stopped = TRUE;
              break;

  case S_RELCF: rc = s_relcf(&scp->sref,&scp->result,&scp->userdata);
              if (rc == S_ERROR) ...       * error handling     *
              scp->state = STA01;
              break;
  case S_GO:    scp->stopped = FALSE;
              if (scp->state == STA713)
                send();
              break;
  case S_TIMEINT:
              if (timeout == TRUE)         * 2nd time interrupt *
                no_event = TRUE;
               else                        * 1st time interrupt *
                {
                timeout = TRUE;
                s_timer(sec);
                }
              break;
  case S_ERROR: errcode = s_error(&addinfo));
              printf("error code %d for s_event-call\n", errcode);
```

```
                 if (errcode == S_TSERR)
                   printf("TS error %d occurred\n",addinfo);
                 exit(-1);
                 break;
    default:      * error handling ...                            *
                 rc = s_uaborq(&sref,NULL);
                 if (scp != NULL)
                   scp->state = STA01;
    }                                          * end switch        *
  }                                            * end while         *
s_detach(&aref);
}                                              * end main          *

send()
  {
  switch(scp->next_action)
    {
    case SENDDATA: strcpy(usdata,"USER DATA...");
                  scp->userdata.len = strlen("USER DATA...");
                  scp->chain = S_END;
                  rc = s_datarq(&scp->sref,usdata,
                                &scp->userdata.len,&scp->chain);
                  if (rc == S_ERROR) ...      * error handling     *
                  scp->next_action = SETSYP;
                  if (rc == S_OK)
                    send();
                   else scp->stopped = TRUE;
                  break;
    case SETSYP  : scp->mtype = S_OPTIONAL;
                  rc = s_minrq(&scp->sref, &scp->mtype,
                               &scp->syncp, NULL, S_END);
                  if (rc == S_ERROR) ...      * error handling     *
                  scp->next_action = (scp->syncp <= 100 ?
                                     SENDDATA : ENDACT);
                  if (rc == S_OK)
                    send();
                   else scp->stopped = TRUE;
                  break;
    case ENDACT  : rc = s_endrq(&scp->sref, &scp->syncp, NULL, S_END);
                  if (rc == S_ERROR) ...      * error handling     *
                  scp->state = STA04B;
                  if (rc == S_STOP)
                    scp->stopped = TRUE;
    }
  }
```

Contents

OSS V3.0 (BS2000/OSD)

OSI Session Service
User Guide

Target group
OSI TP users

Edition: March 1993

File: oss.pdf