Synchronization Indications

When an event associated with synchronization occurs, the `si_type` field of the `si` parameter contains the value `SI_SYNC`, and a `SSAPsync` structure is contained inside the `si` parameter.

```
struct SSAPsync {
    int     sn_type;
#define SN_MAJORIND     0x00
#define SN_MAJORCNF     0x01
#define SN_MINORIND     0x02
#define SN_MINORCNF     0x03
#define SN_RESETIND     0x04
#define SN_RESETCNF     0x05

    int     sn_options;
#define SYNC_CONFIRM    1
#define SYNC_NOCONFIRM  0

#define SYNC_RESTART    0
#define SYNC_ABANDON    1
#define SYNC_SET        2

    long    sn_ssn;
#define SERIAL_NONE     (-1L)
#define SERIAL_MIN      000000L
#define SERIAL_MAX      999998L

    int     sn_settings;

#define SN_SIZE         512
    int     sn_cc;
    char    sn_data[SN_SIZE];
};
```

The elements of a `SSAPsync` structure are:

`sn_type`: defines the synchronization management indication which occurred:

Value	Event
SN_MAJORIND	S-MAJOR-SYNC.INDICATION
SN_MAJORCNF	S-MAJOR-SYNC.CONFIRMATION
SN_MINORIND	S-MINOR-SYNC.INDICATION
SN_MINORCNF	S-MINOR-SYNC.CONFIRMATION
SN_RESETIND	S-RESYNCHRONIZE.INDICATION
SN_RESETCNF	S-RESYNCHRONIZE.CONFIRMATION

`sn_options`: various modifiers of the indication. For the minorsync indication (as described in Section 3.4.4 on page 76):

Value	Modifier
SYNC_CONFIRM	peer wants explicit confirmation
SYNC_NOCONFIRM	peer doesn't want explicit confirmation

For the resync indication (also described in Section 3.4.4):

Value	Modifier
SYNC_RESTART	a "restart" resynchronization is requested
SYNC_ABANDON	a "abandon" resynchronization is requested
SYNC_SET	a "set" resynchronization is requested

`sn_ssn`: the serial-number associated with this synchronization management event;

`sn_settings`: for resync events, the proposed (resync indication) or new (resync confirmation) token settings; and,

`sn_data/sn_cc`: associated data (and the length of that data).

Note that for minorsync events, the user is not obligated to confirm the synchronization point even if the originator requested it.

Activity Indications

When an event associated with activity management occurs, the `si_type` field of the `si` parameter contains the value `SI_ACTIVITY`, and the `si` parameter contains a `SSAPactivity` structure.

```
struct SSAPactivity {
    int     sv_type;
#define SV_START        0x00
#define SV_RESUME       0x01
#define SV_INTRIND      0x02
#define SV_INTRCNF      0x03
#define SV_DISCIND      0x04
#define SV_DISCCNF      0x05
#define SV_ENDIND       0x06
#define SV_ENDCNF       0x07

    struct SSAPactid sv_id;

    struct SSAPactid sv_oid;
    struct SSAPref   sv_connect;

    long    sv_ssn;

    int     sv_reason;

#define SV_SIZE         512
    int     sv_cc;
    char    sv_data[SV_SIZE];
};
```

The elements of a `SSAPactivity` structure are:

`sv_type`: defines the activity management indication which occurred:

Value	Event
SV_START	S-ACTIVITY-START.INDICATION
SV_RESUME	S-ACTIVITY-RESUME.INDICATION
SV_INTRIND	S-ACTIVITY-INTERRUPT.INDICATION
SV_INTRCNF	S-ACTIVITY-INTERRUPT.CONFIRMATION
SV_DISCIND	S-ACTIVITY-DISCARD.INDICATION
SV_DISCCNF	S-ACTIVITY-DISCARD.CONFIRMATION
SV_ENDIND	S-ACTIVITY-END.INDICATION
SV_ENDCNF	S-ACTIVITY-END.CONFIRMATION

`sv_id`: the activity identifier for an activity start or resume indication;

`sv_oid`: the previous activity identifier for an activity resume indication (see page 82);

`sv_connect`: the previous connection identifier for an activity resume indication;

`sv_ssn`: the serial-number for an activity resume or end indication;

`sv_reason`: the reason for an activity interrupt or discard indication (codes are listed in Table 3.5); and,

`sv_data/sv_cc`: associated data (and the length of that data).

Report Indications

When an event associated with exception reporting occurs, the `si_type` field of the `si` parameter contains the value `SI_REPORT`, and a `SSAPreport` structure is contained inside the `si` parameter.

```
struct SSAPreport {
    int     sp_peer;

    int     sp_reason;

#define SP_SIZE         512
```

Provider-initiated Report	
SP_NOREASON	No specific reason stated
SP_PROTOCOL	SS-provider protocol error
User-initiated Report	
SP_NOREASON	No specific reason stated
SP_JEOPARDY	User receiving ability jeopardized
SP_SEQUENCE	User sequence error
SP_LOCAL	Local SS-user error
SP_PROCEDURAL	Unrecoverable procedural error
SP_DEMAND	Demand data token

Table 3.5: SSAP Exception Codes

```
    int     sp_cc;
    char    sp_data[SP_SIZE];
};
```

The elements of a `SSAPreport` structure are:

`sp_peer:` if set, indicates that a user-initiated report occurred (a S-U-EXCEPTION-REPORT.INDICATION event); if not set, indicates that a provider-initiated report occurred (a S-P-EXCEPTION-REPORT.INDICATION event);

`sp_reason:` the reason for the report (codes are listed in Table 3.5 on page 73); and,

`sp_data/sp_cc:` any report data (and the length of that data) from the peer; meaningless if the report is provider-initiated.

Finish Indication

When a S-RELEASE.INDICATION event occurs, the `si_type` field of the `si` parameter contains the value `SI_FINISH`, and a `SSAPfinish` structure is contained inside the `si` parameter.

```
struct SSAPfinish {
#define SF_SIZE         512
    int     sf_cc;
    char    sf_data[SF_SIZE];
};
```

The elements of a `SSAPfinish` structure are:

`sf_data/sf_cc:` any final data (and the length of that data).

3.4.3 Token Management

The fundamental aspect of token management deals with transferring ownership of the tokens.

Sending Tokens

To transfer ownership of one or more session tokens to the remote user, the `SGTokenRequest` routine is called (which corresponds to the S-TOKEN-GIVE.REQUEST action).

```
int     SGTokenRequest (sd, tokens, si)
int     sd;
int     tokens;
struct SSAPindication  *si;
```

The parameters to this procedure are:

`sd:` the session-descriptor;

`tokens:` the tokens to be transferred; and,

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

If the call to `SGTokenRequest` is successful, then ownership of the tokens has been transferred to the remote user.

If activity management has been selected, then the ownership of all tokens can be transferred using the `SGControlRequest` routine (which corresponds to the S-CONTROL-GIVE.REQUEST action).

```
int     SGControlRequest (sd, si)
int     sd;
struct SSAPindication  *si;
```

The parameters to this procedure are:

`sd:` the session-descriptor; and,

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

If the call to `SGControlRequest` is successful, then ownership of all available tokens has been transferred to the remote user. Until this transfer of ownership is acknowledged, other token management functions will (non-fatally) fail.

Requesting Tokens

To request ownership of one or more session tokens, the `SPTokenRequest` routine is called (which corresponds to the S-TOKEN-PLEASE.REQUEST action).

```
int     SPTokenRequest (sd, tokens, data, cc, si)
int     sd;
int     tokens,
        cc;
char   *data;
struct SSAPindication  *si;
```

The parameters to this procedure are:

`sd:` the session-descriptor;

`tokens:` the tokens to requested;

`data/cc:` any additional data (and the length of that data, which may not exceed `ST_SIZE` octets); and,

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

If the call to SPTokenRequest is successful, then the remote user has been notified of the request; however, the ownership of the tokens is not actually transferred until the session provider notifies the user with a S-TOKEN-GIVE-INDICATION event, which typically occurs on the next call to SReadRequest.

3.4.4 Synchronization Management

There are three types of synchronization services: majorsyncs, minorsyncs, and resyncs.

Major Synchronization

To indicate a major synchronization point, the SMajSyncRequest routine is used (which corresponds to the S-MAJOR-SYNC.REQUEST action).

```
int     SMajSyncRequest (sd, ssn, data, cc, si)
int     sd;
long   *ssn;
char   *data;
int     cc;
struct SSAPindication *si;
```

The parameters to this procedure are:

sd: the session-descriptor;

ssn: a pointer to a long integer which, on a successful return, will be updated to reflect the current serial-number ($V(M) - 1$);

data/cc: any additional data (and the length of that data, which may not exceed SN_SIZE octets); and,

si: a pointer to a SSAPindication structure, which is updated only if the call fails.

If the call to SMajSyncRequest is successful, then the major synchronization has been queued for the remote user. When the S-MAJOR-SYNC.CONFIRMATION event is received, the major synchronization is complete. Otherwise the SSAPabort structure contained in the SSAPindication parameter si contains the reason for failure.

Upon receiving a S-MAJOR-SYNC.INDICATION event, the user is required to generate a S-MAJOR-SYNC.RESPONSE action by calling the `SMajSyncResponse` routine.

```
int     SMajSyncResponse (sd, data, cc, si)
int     sd;
char   *data;
int     cc;
struct SSAPindication *si;
```

The parameters to this procedure are:

`sd:` the session-descriptor;

`data/cc:` any additional data (and the length of that data, which may not exceed `SN_SIZE` octets); and,

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

If the call to `SMajSyncResponse` is successful, then the major synchronization has been completed. Otherwise the `SSAPabort` structure contained in the `SSAPindication` parameter `si` contains the reason for failure.

Minor Synchronization

To indicate a minor synchronization point, the `SMinSyncRequest` routine is used (which corresponds to the S-MINOR-SYNC.REQUEST action).

```
int     SMinSyncRequest (sd, type, ssn, data, cc, si)
int     sd;
int     type;
long   *ssn;
char   *data;
int     cc;
struct SSAPindication *si;
```

The parameters to this procedure are:

`sd:` the session-descriptor;

`type`: the type of minor synchronization requested, one of:

Value	Type
`SYNC_CONFIRM`	explicit confirmation requested
`SYNC_NOCONFIRM`	no confirmation requested

`ssn`: a pointer to a long integer which, on a successful return, will be updated to reflect the current serial-number ($V(M) - 1$);

`data/cc`: any additional data (and the length of that data, which may not exceed `SN_SIZE` octets); and,

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

If the call to `SMinSyncRequest` is successful, then the minor synchronization has been queued for the remote user. If a S-MINOR-SYNC.CONFIRMATION event is received, the minor synchronization is complete. However, the remote user is under no obligation to acknowledge the minorsync. Otherwise the `SSAPabort` structure contained in the `SSAPindication` parameter `si` contains the reason for failure.

Upon receiving a S-MINOR-SYNC.INDICATION event, the user may optionally use the `SMinSyncResponse` routine to generate a S-MINOR-SYNC.RESPONSE action.

```
int     SMinSyncResponse (sd, ssn, data, cc, si)
int     sd;
long    ssn;
char   *data;
int     cc;
struct SSAPindication *si;
```

The parameters to this procedure are:

`sd`: the session-descriptor;

`ssn`: the highest serial-number being confirmed;

`data/cc`: any additional data (and the length of that data, which may not exceed `SN_SIZE` octets); and,