Multics Technical Bulletin MTB-531

To: Distribution

From: Richard J.C. Kissel and Kala Marietta

Date: 08/03/81

Subject: Interface Design of the DSA Session Control Implementation on Multics

Send comments to:

Kissel.Multics @ MIT
Kissel.Multics @ System M

or

Richard J.C. Kissel
Honeywell Information Systems
575 Tech. Sq.
Cambridge, Ma. 02139

or

Mail stop: MA22

or call:

617-492-9319
HVN 261-9319

INTRODUCTION

This MTB addresses the interface design for a Multics product which implements the Session Control layer of the Distributed Systems Architecture (DSA). This product will allow Multics to participate in Product Set DSA300, which is the first Honeywell wide supported DSA product. The internal design of the session control product is the subject of another MTB.

SESSION CONTROL OVERVIEW

Session control is located in layer 5 of the Honeywell Distributed Systems Architecture (DSA). It provides services to Presentation control in layer 6 by using the services provided by Transport control in layer 4. The services provided can be divided into: connection establishment, data transfer, and connection termination. The connection establishment service allows initiation of a session to a specified remote endpoint or acceptance of a session from a specified remote endpoint. The data transfer service provides a two-way data path between the local and remote endpoints. The data path can either be two-way alternate, where session control manages which side can send data next, or two-way simultaneous, where session control allows either side to send data at any time. User data enclosures are provided to allow transfer of arbitrarily sized user records, and also, to allow the quarantining of user data before delivery to the correspondent user. An expedited delivery service is provided for certain, short types of user data. The connection termination service allows the session to be terminated in an orderly fashion with no loss of data, or in an abnormal fashion with possible loss of data.

Session control also provides for security on sessions, i.e. user identification and verification, and for administrative control of various aspects of its operation.

SESSION CONTROL ON MULTICS

The following sections provide a detailed description of how session control fits into Multics and how it provides the services outlined above.

General Structure of Session Control

Session control consists of a shared data base, functions which run in users processes, and functions which run in daemon processes. The functions which run in a user's process are provided by an unprivileged gate interface for normal functions, and a privileged gate interface for administrative functions.

The actual work is done in ring 1 because of the need to access and update the shared data base. The functions which run in the daemon process are also done in ring 1.

Transport control runs in a frontend (the UNCP), and the services it provides to session control are available using an interface called the Common Exchange Interface (CXI) over a DIA connection. This interface is timer driven rather than interrupt driven. That is, no interrupts happen between the host and the UNCP; rather, both the host and the UNCP have timers which cause them to periodically check a shared memory area for work to do. The shared memory area is divided into two queues, the input queue and the output queue.

For this reason, as well as a poor fit with the current multiplexer structure, and the amount of processing involved, it was decided to run the "interrupt" side of session control in a daemon process in ring 1. The daemon communicates with the user processes by sending wakeups on user defined event channels when user process work is needed. No communication is needed in the other direction, except as provided by the shared data base. There is one daemon process per UNCP on the system.

The shared data base actually consists of two basic parts. One part is administrative data maintained by a network administrator and used by session control. This part contains data about all of the objects which can be administratively controlled, i.e. frontends, mailboxes, routing tables, etc. The other part is dynamic data maintained by session control to keep track of the operation of logical connections and sessions. This part consists of a segment for logical connection data, a segment for session data, a segment to buffer letters to be sent or received, etc.

The user interface to session control is through a gate into ring 1. This interface permits a session to be: initiated or accepted, normal data to be sent or received, expedited data to be sent or received, negotiated recovery to be done, and normal or abnormal termination to be done. The status of the session may also be checked.

The daemon process has a periodic timer set which causes processing of data in the CXI input queue. This queue contains all of the requests for service from the UNCP. The CXI output queue, which is periodically checked by the UNCP, contains requests for service from Multics, and can be updated by either the daemon process or a user process. The daemon process is responsible for "dummy" traffic which verifies the presence of an operational UNCP, and for taking action if the UNCP does not respond within a certain time period. The daemon process also checks the idle timers associated with session connections, and terminates those connections whose timers have expired.

The administrative interface to session control allows an administrator to define session control objects and update their states. This is also the interface across which session control reports object state changes, and periodically provides status information. This periodic information basically consists of metering information kept by session control. It is stored in an administrative database and can be used for performance measurement.

The security of sessions is enforced by session control using mailbox ACLs (mailbox is used here with a specific session control meaning, not the normal Multics meaning) for local user access control. Remote users are verified using new entries in the Answering Service, and access is checked against DSA access control lists associated with each session. Multics ACLs are not used for DSA access control lists because of differing numbers and lengths of components in the two types of list entries.

More detailed descriptions of the interfaces to the session control implementation on Multics are provided in the following sections. The internal interfaces and internal structure of session control is described in a separate MTB.

The User Interface

The user subroutine interface to session control is described in Appendix A. A normal user will not use this interface directly. Instead, the user will use a presentation control interface and presentation control will use the session control interface. On Multics, the presentation control interface will be through IOX interfaces. For communication with a terminal an I/O module will be provided which implements the terminal management protocol defined for DSA. A record or stream oriented I/O module will also be provided for communication with other processes, for instance, using language I/O. These I/O modules and their interfaces are described in another MTB.

What follows is a basic description of the services provided by session control and a simple scenario for invoking these services through the session control interface.

Session control does not go blocked. Instead, the user supplies up to three event channels over which session control will send wakeups to signal various interesting events. In this respect, session control behaves the same as the ring zero tty dim, which leaves it up to the ring 4 tty_ I/O module to manage blocking. In this case, the presentation control layer would be the user of session control, and would manage blocking.

One event channel, the data input event channel, is used to signal the availability of normal input data on the session. A wakeup will only be sent on this channel if the user has called session_control_$receive and not received any data. When a

wakeup occurs on this channel, the user should call session_control_$receive to get the data that is available.

Another event channel, the data output event channel, is used to signal the ability of session control to accept more output data. A wakeup will be sent on this channel only if the last call to session_control_$send could not send all of the data it was given.

The last event channel, the interrupt event channel, is used to signal the arrival of an interrupt or exceptional event on the session. When a wakeup occurs on this channel, the user should call session_control_$receive_control_info to find out the specific type of interrupt and any associated information.

These three event channels are passed to the initiate and accept_initiates entries by the user as described below. The 72 bits of information passed with a wakeup on these channels will contain an encoded value for the type of event, "input available", "output available", or "interrupt". Therefore, the user could use the same event channel for all three event types and decode the additional information to determine what action to take. Also, if the user uses three event channels, they could be fast event channels, since the encoded value would not be needed.

The establishment service allows a user to initiate a session to a remote correspondent or to accept a session initiated by a remote correspondent. If no appropriate initiate request from a remote correspondent is available, the accept request will be retained by session control and checked against incoming initiate requests until an appropriate one is found. If there is no appropriate accept request to match an incoming initiate request, the initiate request will be retained by session control, for a time specified by the remote user, or until an appropriate accept request is made. The remote correspondent may actually reside on the local system, but this fact is invisible to the user since session control provides a routing function as part of the establishment service.

The user first finds an appropriate Session Type Descriptor (STD) to describe the characteristics of the session he would like to establish. In general, STDs will be built by administrators and used "as is" by users, however, the user is free to create his own, or to copy an existing one and modify it to suit his particular requirements. An STD is identified by a pathname which will be resolved by using the session_control search list. The archive component pathname convention will be respected. If the local or remote mailboxes (session endpoints) are either not specified in the STD, or not acceptable, the user may change them in the establishment call. If any presentation control or user data is to be sent with an initiate request, it must be supplied by the user since it is not in the STD. The user also creates the necessary event channels as described above.

Having obtained the necessary inputs, the user will then call the session_control_$initiate entry if he wants to initiate a session, or the session_control_$accept_initiates entry if he wants to accept a session. After checking the supplied arguments and verifying security parameters, session control will start the establishment of the session and return to the user.

In the initiate case, when an appropriate acceptance of an initiate request is received from the remote correspondent, session control will send a wakeup on the interrupt event channel that the user specified in the initiate call. The user should then call the session_control_$receive_control_info entry to receive any data the remote correspondent sent with his acceptance. At this point the session is established and can be used as described below. The final negotiated characteristics of the session (i.e. based on the characteristics supplied in the STD as negotiated with the remote correspondent) are returned from the call to the session_control_$receive_control_info entry.

In the accept case, when an appropriate initiate request is received from a remote correspondent, session control will send a wakeup on the interrupt event channel that the user specified in the accept_initiates call. The user should then call the session_control_$receive_control_info entry to receive any data the remote correspondent sent with his initiate request. The user will also get the negotiated characteristics of the session. The user can then make a decision as to whether the session is acceptable. He then calls the session_control_$ack_initiate entry to either accept or reject the session (the reason code indicates acceptance or rejection), and to send any presentation or user data with the acknowledgement. At this point, the session is established and can be used as described below.

On an established session, session control provides a number of services to aid the two correspondents in structuring their dialogue.

A normal data transfer service allows data to be sent and received. The data may be delimited by "end of record" or "end of quarantine unit" enclosures (if quarantining is allowed on the session) that are set by the sending user and visible to the receiving user. The enclosure level, and any additional information such as record type, is carried with the data, but returned to the user in a separate structure. If quarantining is allowed on the session, the receiver will not see any part of a quarantine unit until the sender closes that quarantine unit. Closing a quarantine unit also closes the current record. If quarantining is allowed on the session, the sender may cancel the record or quarantine unit he is currently sending. This is invisible to the receiver. If quarantining is not allowed on the session, the receiver can receive data at any time and will simply receive an indication when "end of record" occurs. If session control cannot accept all of the data specified in a send

request, a wakeup will be sent on the data output event channel when more data can be accepted.

On two-way alternate sessions a "turn" management service is also provided for normal data transfer. Session control will only allow the correspondent with the "turn" to send data and the correspondent without the "turn" to receive data. This service is provided by another level of enclosure called "end of interaction unit". This enclosure level may only be set by the correspondent with the "turn", and it causes the "turn" to be sent to the other correspondent. Closing the interaction unit also closes the current quarantine unit (if any) and the current record. The "end of interaction unit" enclosure may also be used on two-way simultaneous sessions (with the same hierarchical closure property), but session control will not enforce the use of the "turn".

The use of quarantining, and the mode of the session (two-way alternate or two-way simultaneous), are negotiated during session establishment. Receipt of normal data will be indicated, if necessary, by a wakeup sent on the data event channel that was supplied by the user when the session was established. The normal data transfer services are invoked by using the session_control_$send, session_control_$receive. If the cancellation service is allowed, it is invoked by using the session_control_$cancel entry.

Session control also provides an expedited, or interrupt, data transfer service. This service allows either correspondent to send either 16 bits of data (the attention interrupt) or up to 140 characters of data (the data_attention interrupt). Data sent in this manner will bypass any normal data session control may be holding for the user; therefore, this data will be delivered out of sequence with any normal data. Receipt of interrupt data will be indicated by a wakeup sent on the interrupt event channel that was supplied by the user when the session was established. Use of the data_attention service is negotiated during session establishment. The attention service is invoked by using the session_control_$attention entry, and the data attention service is invoked by using the session_control_$data_attention entry. Both interrupts are received by using the session_control_$receive_control_info entry.

Session control provides a "demand turn" service on two-way alternate sessions which allows the correspondent without the "turn" to take the turn away from the other correspondent. The correspondent demanding the turn may send 16 bits of data to the other correspondent to inform him of the reason for this action. Any normal data which session control may be holding for either correspondent will be deleted when this service is requested. This service is not allowed if the marking or recovery services described below have been negotiated for use on the session. Receipt of a demand turn request causes session control to send a wakeup on the interrupt event channel of the receiving

correspondent. This service is invoked by the session_control_$demand_turn entry. The receipt of the "turn" will be indicated by an empty data record which has an "end of interaction unit" enclosure specified. This record is received by using the session_control_$receive entry.

Session control provides a "purge" service on sessions for which the marking or recovery services described below have not been negotiated. This service allows the sender of normal data to purge any of that data which session control has not yet delivered to the receiver. Notice that this may or may not delete any data, depending on how much data the receiver has already gotten, therefore, the sender does not know how much data (if any) has been deleted. The sender may also send 16 bits of data with this request to indicate to the receiver the reason for this action. Receipt of a purge request causes session control to send a wakeup on the interrupt event channel of the receiving correspondent. This service is invoked by the session_control_$purge entry. It is received by using the session_control_$receive_control_info entry.

Session control provides a marking and recovery service for the normal data flow between two correspondents. Using the marking service, each correspondent can put a mark in the data he sends, and have the other correspondent receive that mark. Session control counts these marks so that each mark can be identified by its sequence number. The sequence number of a mark is returned to the receiver when a marked record is received. Notice that the marks in the two directions of normal data flow are independent. Each mark delimits a Session Recovery Unit (SRU), so named because the marks are used in the recovery service described below. The marking service is invoked by using the "end of sru" enclosure when sending normal data. The "end of sru" enclosure is independent of the other enclosures described above for the normal data flow and may be sent at any time without affecting the other enclosures. Each mark must be acknowledged by the receiver; however, a group of consecutive marks may be acknowledged in one operation. Marks are acknowledged by using the session_control_$release_sru entry. Receipt of the acknowledgement will be indicated by a wakeup sent on the interrupt event channel. The user will then use the session_control_$receive_control_info entry to get the sequence number of the mark being acknowledged. The use of the marking service is negotiated during session establishment.

The recovery service allows the two correspondents to negotiate a recovery point in their dialogue based on the previously agreed upon marks defined by the marking service. This service also allows the correspondents to negotiate the position of the "turn" after the recovery. Session control will also purge any normal data (and interrupt data if requested) which is being held for either correspondent if the data is outside the scope agreed to in the recovery negotiation.

Even if the marking service is not used, the recovery service may still be used to purge all unreceived data in both directions by specifying a "0" mark for both directions of data flow in the recovery request.

The recovery service can be invoked by either correspondent at any time by using the session_control_$recover entry. The remote correspondent is notified of the request by a wakeup on the interrupt event channel. The user will then use the session_control_$receive_control_info entry to get the recovery information. He then uses the session_control_$recover_ack entry to send an acknowledgement, and the original requestor is notified of the acknowledgement in the same way. At this point both correspondents have agreed on the recovery point, and session control has performed any necessary actions, so the session is back in a normal state. The use of the recovery service is negotiated at session establishment time.

Session control provides a "suspension" service. This service may be used by either correspondent at any time to suspend the normal data flow on a session. All other session services are still available on a suspended session (e.g. the interrupt service, the recovery service, etc.). The correspondent requesting the suspension may send 16 bits of data to indicate the reason for the suspension. The correspondent that suspends a session must also resume it. Notice that both correspondents may suspend the session at the same time, in which case both correspondents must resume the session before it is in a normal state. This service is invoked by using the session_control_$suspend and session_control_$resume entries. No action is required of the receiving correspondent, since session control take all the necessary actions to respond to these requests. However, the receiving correspondent is notified of the request by a wakeup on the interrupt event channel. The subsequent session_control_$receive_control_info will indicate "suspend" or "resume" respectively. If the receiving correspondent attempts to use the normal data transfer service while the session is suspended, an error code will be returned. The sender of the suspend or resume requests is notified of the acknowledgement to these requests by a wakeup on the interrupt event channel. The subsequent session_control_$receive_control_info will indicate "suspend_ack" or resume_ack" respectively.

Finally, session control provides normal and abnormal termination services. The abnormal termination service may be used at any time by either correspondent. This service allows the initiator of the service to send 16 bits of data to the remote correspondent to indicate the reason for the termination. This service destroys any data which is held by session control for either correspondent. The remote correspondent is notified of this action by a wakeup on the interrupt event channel. The subsequent session_control_$receive_control_info will indicate "terminate_abnormal". There is no user visible acknowledgement

for this service, it takes effect immediately. This service is invoked by using the session_control_$terminate_abnormal entry.

Normal termination does not destroy data and allows both correspondents to terminate the session gracefully. Normal termination allows the two directions of data flow to be terminated independently. The service is invoked by using the "end of session" enclosure on the normal data flow (by setting the enclosure level in the session_control_$send call). This enclosure also closes the current interaction unit, the current quarantine unit (if any), and the current record. It is also a mark, like any other "end of sru" mark. The sender should continue to receive normal data from the remote correspondent until he receives an "end of session" enclosure (by using session_control_$receive). At this point the session has been terminated normally, and is no longer available for any services. After a correspondent has sent "end of session", and before he has received "end of session", he may only use the normal and interrupt data receive services, the recovery service, or the terminate abnormal service.

The Administrative Interface

The administrative subroutine interface to session control is described in Appendix B. The interface is through a gate and allows a Network Adminstrator to modify the session control database as well as find out status information about particular session control objects. This interface also provides the mechanism for creating the initial configuration of session control on a system, and for initializing session control operations at system startup time.

This interface will probably only be used by the Network Administration (NAD) function defined by DSA Distributed Systems Administration and Control (DSAC). The implementation of the NAD, and the interface to it, is described in another MTB. The NAD will also provide an interface for session control to use to report status and metering information.

Session Security

In DSA, a session is a connection between two endpoints called mailboxes. One mailbox is used by the initiator of the session and is local to the system on which the initiator is running. The other mailbox may either be on the same system (an internal session) or on a remote system. A mailbox may be the endpoint for many sessions at the same time. It may also have some special properties, e.g. a process is created when an incoming initiate session request is received. A mailbox is identified by an 8 character name and a 4 character extension. The extension provides the capability to address a particular correspondent in a group of correspondents all using the same

mailbox. For instance, the separate printers in a pool managed by a printer manager could be addressed by using the mailbox of the printer manager with the extension specifying a particular printer. The concatenation of session control name, mailbox name, and mailbox extension is called a Standard Global Name (SGN) and must be unique within the entire network.

A session control mailbox on Multics consists of an entry in an administrative database. This entry will be found by using the mailbox name. For the purpose of security, this entry will contain the pathname of an ACS segment for the mailbox. The ACL on this ACS segment is used as described below.

In order to ensure the security of sessions on Multics, two sorts of access control are proposed. The local aspect of security will be implemented by using ACLs on local mailboxes. The "rew" control of the ACL will be used in the following way. A process is required to have "r" access to the mailbox in order to accept sessions on that mailbox. A process is required to have "w" access to the mailbox in order to initiate sessions on that mailbox. And, a process is required to have "e" access to the mailbox to perform administrative functions on the mailbox, e.g. find out how many sessions are using the mailbox, find out the identity of the processes with sessions using the mailbox, delete the mailbox, etc.

The remote aspect of security will be implemented by using access control lists in the session type descriptor (STD). The STD is a structure passed to session control by the user when a session is to be accepted or initiated. It contains all the information necessary for session control to establish the session, e.g. two-way alternate or two-way simultaneous, quarantine unit size, security information, etc. The access control lists in the STD are called DSA remote ACLs. They are different from normal Multics ACLs because they contain identification information in a different format from a Multics ACL. They are defined as a standard security record in DSA and contain a person, project, and billing identifier, each 12 characters long. In the STD, the DSA remote ACL specifies two types of allowed remote access. The first type is whether the specified remote user is allowed to initiate a session which will be accepted by the local user. The second type is whether the specified remote user is allowed to accept a session initiated by the local user.

The local aspect of security will be enforced by session control when the local user calls to initiate or accept a session. The remote aspect of security will be enforced when a remote initiate or initiate acknowledge (accept) request is received by session control.

We now discuss how the remote user identity (person, project, and billing) is verified by Multics session control. The Submitter Identity Record sent when a session is initiated or

accepted contains a password field, 96 bits (8 characters), as well as the previously mentioned person, project, and billing fields. Session control will call the answering service at a special entry with this information and receive back a "verified" or "not verified" indication. The answering service will maintain the user identity/password correspondence for remote users in a way similar to the way this correspondence is maintained currently for remote batch users and the card input password. The current concept will be extended to allow remote access through DSA networks, so the password will become a remote access password. This method of user validation will lead to low security access to Multics. That is, a user who logs in this way will have his access to Multics restricted in some way. This will probably be done with a new tag character on the user_id to indicate that the user was validated by the remote validation process. Possibly AIM could be used to restrict access, but the implications of this route are not currently clear.

For high security access to Multics (the current normal sort of access), user validation will be done differently. This will involve the Answering Service accepting any session initiated to the "Login" mailbox, without checking any DSA defined security parameters. The Answering Service will then engage in the normal Multics login dialogue with the user of the session. If the user is validated, his process will be informed of the session_id to use when talking to the terminal. This is analagous to the way logins over tty channels and ARPANET sockets are currently handled.

Both levels of access to Multics are described more fully in another MTB.

RELATED TOPICS

These topics will be covered in future MTBs.

- Internal Design of the DSA Session Control Implementation on Multics (MTB-532)
- The DSA DSAC Implementation on Multics
- Answering Service Changes for Logins Over DSA Sessions
- DSA Presentation Control and Terminal Management
- DSA Security Control on Multics

APPENDIX A

This appendix specifies the user gate interface which will be provided to session control. In general, it is expected that this interface will be used by a presentation control layer; however, applications which do not require presentation layer services may use it directly.

Name: session_control_

This gate is the user interface to session control on Multics. The entries are described in the following sections. The entries available are:

session_control_$accept_initiates
session_control_$ack_initiate
session_control_$attention
session_control_$cancel
session_control_$data_attention
session_control_$demand_release_sru
session_control_$demand_turn
session_control_$initiate
session_control_$purge
session_control_$receive
session_control_$receive_control_info
session_control_$recover
session_control_$recover_ack
session_control_$release_sru
session_control_$resume
session_control_$send
session_control_$status
session_control_$suspend
session_control_$terminate_abnormal

Entry: session_control_$accept_initiates

This entry allows the caller to accept a session on a specified local mailbox, from a specified remote address, with specified parameters. The user will receive a wakeup on the interrupt event channel when a matching initiate request from a remote user is received. The session_control_$receive_control_info entry should then be called to receive any presentation control data associated with the initiate request. The user may then accept or reject the connection using the session_control_$ack_initiate entry.

Usage

```
dcl session_control_$accept_initiates entry
        (ptr, ptr, char (*), (*) fixed bin (71),
                fixed bin (16), fixed bin (35));

call session_control_$accept_initiates
        (local_sgnp, remote_sgnp, std_name, event_channels,
                session_id, code);
```

where:

local_sgnp (Input)
is a pointer to a structure which contains the standard global name (SGN) of the the local endpoint for the session. The structure is described below.

remote_sgnp (Input)
is a pointer to a structure which contains the SGN of the remote endpoint for the session. The structure is described below.

std_name (Input)
is the pathname of a session type descriptor (STD) structure describing the parameters of the desired session. The name will be resolved into a pointer by using the session_control search paths. The structure may reside in a segment or an archive. The structure is described below.

event_channels (Input)
is an array containing up to three event channels which session control is to use to notify the user of special occurrences.

The first event channel is the data input channel. A wakeup is sent on this channel when there is input data available and the previous receive operation resulted in no data being returned.

The second event channel is the data output channel. A wakeup is sent on this channel when more data can be accepted after a send operation has failed to send all of the specified data.

The third event channel is the interrupt event channel. A wakeup is sent on this channel when any of a number of special session control events occurs.

The information sent with the wakeups on these event channels will contain an encoded value that indicates which type of event it is. The encoding is described below. This allows one or two event channels to be used, rather than all three. Multiple use of the same event channel may be specified either by putting the same event channel in more than one of the array elements, or by passing an array of fewer than three elements. The event channel in the first element is used in place of any missing elements. Any of these event channels may be fast event channels. In this case, three separate channels should be specified, since no wakeup information can be sent.

session_id (Output)
is the session identifier which is to be used in all subsequent calls to session_control_ entries. It is unique within the local system.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Notes

The event channel information has the following encoding (declared in session_dcls.incl.pl1):

```
dcl 1 event_message          aligned based,
      2 wakeup_name          char (4) unaligned,
      2 pad                  bit (27) unaligned,
      2 interrupt_type       fixed bin (9) unsigned unaligned;
```

where:

wakeup_name
is "recv" for the data input wakeup, "send" for the data output wakeup, and "intr" for the interrupt wakeup.

interrupt_type
is only valid with the "intr" wakeup_name. The values are given under the session_control_$receive_control_info entry.

The local_sgnp and remote_sgnp point to a structure of the following form (declared in session_dcls.incl.pl1):

```
dcl 1 session_control_sgn      aligned based,
      2 session_control_id     char (4) unaligned,
      2 mailbox                char (8) unaligned,
      2 extension              char (4) unaligned;
```

where:

session_control_id
is the name assigned to the session control which contains the mailbox. This may be all blanks if the mailbox belongs to the local session control.

mailbox
is the name of a mailbox which is to be used as a session control addressing point. If this is all blanks, then the mailbox name specified in the session_type_descriptor structure will be used. It is an error if both mailbox names are all blanks.

extension
is the mailbox extension which is to be used as a session control addressing point. This may be all blanks if the name is unknown, or no extension is to be used.

The std_name is used to find a structure with the following form (declared in session_dcls.incl.pl1):

```
dcl 1 session_type_descriptor aligned based,
      2 version               fixed bin (35),
      2 local_sgn             aligned like session_control_sgn,
```

```
    2 remote_sgn              aligned like session_control_sgn,
    2 logical_connection      aligned,
     3 quarantining           aligned,
      4 sending_cap           fixed bin (4) unsigned unaligned,
      4 receiving_cap         fixed bin (4) unsigned unaligned,
      4 pad1                  bit (28) unaligned,
     3 type                   aligned,
      4 twa_initiator_turn    bit (1) unaligned,
      4 twa_acceptor_turn     bit (1) unaligned,
      4 tws_initiator_turn    bit (1) unaligned,
      4 tws_acceptor_turn     bit (1) unaligned,
      4 one_way               bit (1) unaligned,
      4 pad2                  bit (31) unaligned,
     3 protocols              aligned,
      4 suspend               bit (1) unaligned,
      4 recover               bit (1) unaligned,
      4 resynchronize         bit (2) unaligned,
      4 data_attention        bit (1) unaligned,
      4 sru_correlation       bit (1) unaligned,
      4 demand_release_sru    bit (1) unaligned,
      4 commit_support        bit (1) unaligned,
      4 init_superior         bit (1) unaligned,
      4 init_end_point        bit (1) unaligned,
      4 pad3                  bit (26) unaligned,
     3 options                aligned,
      4 nine_bit_bytes        bit (1) unaligned,
      4 request_security      bit (1) unaligned,
      4 max_subchannels       fixed bin (8) unsigned unaligned,
      4 init_send_max_sru     fixed bin (4) unsigned unaligned,
      4 accept_send_max_sru   fixed bin (4) unsigned unaligned,
      4 pad4                  bit (18) unaligned,
    2 submitter_id            aligned,
     3 person                 char (12) unaligned,
     3 project                char (12) unaligned,
     3 billing                char (12) unaligned,
     3 password               char (8) unaligned,
    2 terminal_id             aligned,
     3 terminal               char (24) unaligned,
    2 num_ra                  fixed bin,
    2 remote_acl              (0 refer num_ra) aligned
                              like dsa_remote_acl_entry;
```

where:

version
: is the version of this structure. It should be set to session_type_descriptor_v1.

local_sgn
specifies the name of the local endpoint of the session. When combined with the value specified as an argument to the session_control_$accept_initiates or session_control_$initiate entries, all of the fields must be fully resolved, otherwise an error will be returned. If corresponding fields in both the subroutine argument and the session_type_descriptor structure are specified, then the one in the subroutine argument is used.

remote_sgn
specifies the name of the remote endpoint of the session. When combined with the value specified as an argument to the session_control_$accept_initiates or session_control_$initiate entries, all of the fields must be fully resolved, otherwise an error will be returned. If corresponding fields in both the subroutine argument and the session_type_descriptor structure are specified, then the one in the subroutine argument is used.

sending_cap
specifies whether or not quarantining should be done in the sending direction, and if so, what the quarantining capacity (maximum quarantine unit size) is. A value of 0 requests the correspondent to do quarantining. A value of 15 indicates that quarantining should not be done in the sending direction. Other values are interpreted as 2**(n + 4) bytes.

receiving_cap
specifies the quarantining capacity in the receiving direction. It is interpreted the same way as sending_cap. Note that 15 is not a valid value.

twa_initiator_turn
indicates the user's desire for a two way alternate session with the initiator of the session having the turn first. If this is specified with other type options, then one will be picked from the following ordered list:

tws initiator turn
tws acceptor turn
twa initiator turn
twa acceptor turn
one-way

twa_acceptor_turn
indicates the user's desire for a two way alternate session with the acceptor of the session having the turn first. If this is specified with other type options, then one will be picked as described above.

tws_initiator_turn
indicates the user's desire for a two way simultaneous session with the initiator of the session having the turn first. If this is specified with other type options, then one will be picked as described above.

tws_acceptor_turn
indicates the user's desire for a two way simultaneous session with the acceptor of the session having the turn first. If this is specified with other type options, then an appropriate one will be picked.

one_way
indicates the user's desire for a one-way session. If this is specified with other type options, then one will be picked as described above.

suspend
indicates the user's desire to have the suspend protocol supported.

recover
indicates the user's desire to have the recover protocol supported.

resynchronize
indicates the user's desire to have the resynchronize protocol supported.

data_attention
indicates that the user expects to use data attention records.

sru_correlation
indicates that the user expects to use the SRU correlation request and acknowledge facility.

demand_release_sru
indicates that the user expects to use the demand_release_sru record.

commit_support
indicates that the user expects commitment to be supported on the session.

init_superior
indicates that the user expects the initiator of the session to be superior in the commitment protocol. This is only valid if commit_support = "1"b.

init_end_point
indicates that the user expects the initiator of the session to be an end-point in the commitment protocol. This is only valid if commit_support ="1"b.

nine_bit_bytes
indicates that the user desires to be able to send binary as well as character data on the session.

request_security
indicates that the user wants security information from the correspondent.

max_subchannels
indicates the maximum number of subchannels to be allowed on a logical connection.

init_send_max_sru
specifies the maximum number of unreleased session recovery units the user is willing to have in his sending direction of data flow.
0 - no use of sru's will be made
1 - 14 2**(n-1) unreleased srus
15 - unlimited number of unreleased srus

accept_send_max_sru
specifies the maximum number of unreleased session recovery units the acceptor should allow in his sending direction of data flow. The number is interpreted in the same way as init_send_max_sru.

pad
is reserved and must be zero.

person
is the DSA person id to be used in establishing this session. If this is all blanks, then the user's current Multics person_id will be used (truncated to 12 characters if necessary).

project
is the DSA project id to be used in establishing this session. If this is all blanks, then the user's current Multics project_id will be used.

billing
is the billing id to be used in establishing this session. If this is all blanks, then no billing id will be used.

password
is the password to be used in establishing this session. If this is all blanks, then no password is assumed.

terminal
is a terminal identifier that should be used in establishing this session. If this is all blanks, then no terminal identifier is assumed.

remote_acl
is a list of dsa acl entries which should be checked against the identification of a remote initiator or acceptor. The remote initiate or accept will be rejected if the remote user does not have the proper access. The structure of an entry in this list is described below.

The structure of a dsa_remote_acl_entry is as follows (declared in session_dcls.incl.pl1):

```
dcl 1 dsa_remote_acl_entry     aligned based,
      2 id                     aligned,
        3 person               char (12) unaligned,
        3 project              char (12) unaligned,
        3 billing              char (12) unaligned,
      2 access                 aligned,
        3 allow_remote_init    bit (1) unaligned,
        3 allow_remote_accept  bit (1) unaligned,
        3 pad                  bit (34) unaligned;
```

where:

person
is the person name of the remote user. It may be "*" meaning any person name.

project
is the project name of the remote user. It may be "*" meaning any project name.

billing
is the billing name of the remote user. It may be "*" meaning any billing name.

allow_remote_init
is the access mode which allows an initiate request from the specified remote user to be matched with this request. Note that this may be further restricted by the remote acl on the specified local mailbox.

allow_remote_accept
is the access mode which allows the acceptance of an initiate by the specified remote user to be valid. Note that this may be further restricted by the remote acl on the specified local mailbox.

Entry: session_control_$ack_initiate

This entry allows the caller to indicate acceptance or rejection of a session after an initiate arrives and the caller has processed any establishment data that was present.

Usage

```
dcl session_control_$ack_initiate entry
          (fixed bin (16), ptr, fixed bin (16), fixed bin (35));

call session_control_$ack_initiate
          (session_id, pc_data_ptr, reason, code);
```

where:

session_id (Input)
is the session identifier of the session to be accepted or rejected.

pc_data_ptr (Input)
is a pointer to a structure which describes the presentation control data to be sent to the initiator. The structure is described below.

reason (Input)
is a reason code describing whether the session is accepted or rejected. If the code is 0, the session is accepted. If it is non-zero, the session is rejected. The reason codes for this, and all the other entries, are defined in DSA55 Section 5.5.12.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Notes

The pc_data structure pointed to by pc_data_ptr is described below (a declaration can be found in session_dcls.incl.pl1):

```
dcl pc_data_number_of_records          fixed bin;

dcl 1 pc_data                  aligned based,
    2 version                  fixed bin (35),
```

```
     2 number_of_records       fixed bin,
     2 record_array            (pc_data_number_of_records
                                refer pc_data.number_of_records),
       3 info                  aligned like record_info,
       3 record_ptr            ptr,
       3 record_len            fixed bin (14);
```

where:

version
: is the version of this structure. It should be set to pc_data_v1.

number_of_records
: is the number of records described by this structure.

record_array
: is an array which describes each record and contains a pointer to the record data.

info
: is a structure describing the record. The record_info structure is described below.

record_ptr
: is a pointer to the data for the record.

record_len
: is the length of the record in 9-bit bytes.

The record_info structure is described as follows (a declaration can be found in session_dcls.incl.pl1):

```
dcl 1 record_info              aligned based (recip),
      2 version                fixed bin (35),
      2 type                   fixed bin (12) unsigned,unaligned,
      2 enclosure              fixed bin (17) unaligned,
      2 pad                    bit (6) unaligned,
      2 sru_number             fixed bin (32) aligned,
      2 flags                  aligned,
        3 ascii                bit(1) unaligned,
        3 control              bit(1) unaligned,
        3 correlation_request  bit(1) unaligned,
        3 correlation_ack      bit(1) unaligned,
        3 pad                  bit(32) unaligned;
```

where:

version
: is the version of this structure. It should be set to record_info_v1.

type
: is the record type associated with the record. Currently, this must be in the range: 0 to 3999 if the control bit is "0"b (4000 to 4095 are reserved), or in the range: 128 to 4095 if the control bit is "1"b (0 to 127 are reserved for session control). The record types are defined in the various DSA protocol documents and in the Unified File Format document.

enclosure
: specifies the session enclosure associated with this record. See the description of the possible enclosures below.

sru_number
: specifies the SRU number for this record. It is only valid if enclosure indicates an end of sru enclosure. If this structure describes a record to be sent, this value is ignored.

ascii
: is the data type of the record. If "1"b, the record consists of character data (8 bits of data per 9 bit byte). If "0"b, the record consists of binary data (9 bits of data per 9 bit byte).

control
: is set to indicate whether or not this record is a control record.

correlation_request
: is set to indicate whether or not this record should be correlated with another record. This bit is only valid if an end of SRU enclosure is also present.

correlation_ack
: is set to indicate whether or not this record is a response to a correlation request. This bit is only valid if an end of SRU enclosure is also present.

The possible enclosures and their values are as follows: (declarations can be found in session_dcls.incl.pl1)

Enclosure	Value
no enclosure	1
end of quarantine unit	2
end of interaction unit	3
end of committment unit	4
end of sru	5
end of quarantine unit and end of sru	6
end of interaction unit and end of sru	7
end of session	8
end of record	9
end of record and end of sru	10

Entry: session_control_$attention

This entry allows the caller to send an attention request on the specified session.

Usage

```
dcl session_control_$attention entry (fixed bin (16),
                        fixed bin (16) unsigned, fixed bin (35));

call session_control_$attention (session_id,
                        reason, code);
```

where:

session_id (Input)
is the session identifier of the session on which the request is to be sent. This is returned by the initiate or accept_initiates entries.

reason (Input)
is the reason code to be sent with the request.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$cancel

This entry allows the caller to cancel either the current record or quarantine unit. This is only possible if quarantining has been negotiated on the session.

Usage

```
dcl session_control_$cancel entry (fixed bin (16), bit (1),
                        fixed bin (35));

call session_control_$cancel (session_id, record,
                        code);
```

where:

session_id (Input)
is the session identifier of the session on which a record or quarantine unit is to be cancelled. It is returned by the initiate or accept_initiates entries.

record (Input)
indicates whether a record or a quarantine unit is to be cancelled.
"1"b cancel a record
"0"b cancel a quarantine unit

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$data_attention

This entry allows the caller to send a data attention request on the specified session.

Usage

```
dcl session_control_$data_attention entry (fixed bin (16),
                        char (*) varying, fixed bin (35));

call session_control_$data_attention (session_id,
                        attn_data, code);
```

where:

session_id (Input)
is the session identifier of the session on which the request is to be sent. This is returned by the initiate or accept_initiates entries.

attn_data (Input)
is the data to be sent with this request. The maximum length of the data is 140 characters.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$demand_release_sru

This entry allows the caller to demand the receiver to release a specified SRU on the sender's sending data flow.

Usage

```
dcl session_control_$demand_release_sru entry (fixed bin (16),
                        fixed bin (32), fixed bin (35));

call session_control_$demand_release_sru (session_id,
                        sru_number, code);
```

where:

session_id (Input)
is the session identifier of the session on which to release the sru. This is returned by the initiate or accept_initiates entries.

sru_number (Input)
is the number of the sru to be released. Notice that this applies to the sender's sending data flow.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$demand_turn

This entry allows the caller to send a demand turn request on the specified session. This request may only be used on two-way alternate sessions for which the use of recovery and SRUs has not been negotiated.

Usage

```
dcl session_control_$demand_turn entry (fixed bin (16),
                        fixed bin (16) unsigned, fixed bin (35));

call session_control_$demand_turn (session_id,
                        reason, code);
```

where:

session_id (Input)
is the session identifier of the session on which the request is to be sent. This is returned by the initiate or accept_initiates entries.

reason (Input)
is the reason code to be sent with the request.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$initiate

This entry allows the caller to initiate a session from a specified local mailbox to a specified remote mailbox, with specified parameters. When a valid acceptance is received, the user will receive a wakeup on the interrupt event channel. He may then receive any data returned by the remote acceptor by using the session_control_$receive_control_info entry. The session may then be used or may be abnormally terminated if the acceptance data is not acceptable.

Usage

```
dcl session_control_$initiate entry
          (ptr, ptr, char (*), ptr, (*) fixed bin (71),
                fixed bin (16), fixed bin (35));

call session_control_$initiate
          (local_sgnp, remote_sgnp, std_name, pc_data_ptr,
                event_channels, session_id, code);
```

where:

local_sgnp (Input)
is a pointer to a structure which contains the standard global name (SGN) of the local *endpoint* for the session. The structure is described in the session_control_$accept_initiates entry.

remote_sgnp (Input)
is a pointer to a structure which contains the SGN of the remote endpoint for the session. The structure is described in the session_control_$accept_initates entry.

std_name (Input)
is the pathname of a session type descriptor (STD) structure describing the parameters of the desired session. The name will be resolved into a pointer by using the session_control search paths. The structure may reside in a segment or an archive. The structure is described in the session_control_$accept_initates entry.

pc_data_ptr (Input)
is a pointer to a structure which describes the

presentation control data which should be included in the initiate request. Session control includes this data without interpretation. See the session_control_$ack_initiate entry for a description of the pc_data structure.

event_channels (Input)
is an array of up to three event channels which session control is to use to notify the user of special occurrences. See the accept_initiates entrypoint for a complete description.

session_id (Output)
is the session identifier which is to be used in all subsequent calls to session_control_ entries. It is unique within the local system.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$purge

This entry allows the caller to send a purge request on the specified session. This request may not be sent on a session for which recovery or the use of SRUs has been negotiated.

Usage

```
dcl session_control_$purge entry (fixed bin (16),
                          fixed bin (16) unsigned, fixed bin (35));

call session_control_$purge (session_id,
                          reason, code);
```

where:

session_id (Input)
is the session identifier of the session on which the request is to be sent. This is returned by the initiate or accept_initiates entries.

reason (Input)
is the reason code to be sent with the request.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$receive

This entry returns any data (up to and including the next data record) on the session and the associated record information.

Usage

```
dcl session_control_$receive entry (fixed bin (16), ptr,
                        fixed bin (21), ptr,
                        fixed bin (21), fixed bin (35));

call session_control_$receive (session_id, buf_ptr,
                        buf_len, record_info_ptr
                        bytes_rcvd, code);
```

where:

session_id (Input)
is the session identifier on which data is to be received. This is returned by the initiate or accept_initiates entries.

buf_ptr (Input)
is a pointer to a buffer in which the data should be returned.

buf_len (Input)
is the length of the data buffer in 9-bit bytes.

record_info_ptr (Input)
is a pointer to a structure which describes the received record or data. The structure is filled in by session control. The record_info structure is described in the session_control_$ack_initiate entry.

bytes_rcvd (Output)
is the number of bytes actually returned by this receive. If this value is zero, then a wakeup will be sent on the data input event channel when data is available.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$receive_control_info

This entry allows the caller to receive any of various session control interrupts and any associated data on the specified session.

Usage

```
dcl session_control_$receive_control_info entry (fixed bin (16),
                        fixed bin (17), area, ptr,
                        fixed bin (35));

call session_control_$receive_control_info (session_id,
                        interrupt_type, info_area, info_ptr,
                        code);
```

where:

session_id (Input)
is the session identifier of the session on which the interrupt is to be received. This is returned by the initiate or accept_initiates entries.

interrupt_type (Output)
is the type of interrupt received. Values for the interrupt types are given below.

info_area (Input)
is an area in which additional information associated with the interrupt can be allocated.

info_ptr (Output)
is a pointer to any additional information about the interrupt. This may be null if no additional information is returned. The additional information is allocated in info_area. The particular structure of the additional information depends on interrupt_type. The various structures are described below.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Notes

The possible interrupt types and their values are as follows (declarations can be found in session_dcls.incl.pl1):

Enclosure Type	Value
attention	1
data_attention	2
demand_turn	3
purge	4
term_abnormal	5
recovery	6
release_sru	7
suspend	8
resume	9
suspend_ack	10
resume_ack	11
establishment	12

For the following interrupt types:

attention
purge
term_abnormal
resume

info_ptr points to the following structure:

```
dcl reason          bit (16) aligned based;
```

For the following interrupt type:

demand_turn

info_ptr points to the following structure:

```
dcl 1 demand_turn_info          aligned based,
      2 reason                  bit (16),
      2 enclosure_rcvd          fixed bin (17);
```

For the following interrupt type:

data_attention

info_ptr points to the following structure:

```
dcl attn_info       char (140) varying aligned based;
```

For the following interrupt type:

recovery

info_ptr points to the recovery_info structure as declared in session_control_$recover.

For the following interrupt type:

release_sru

info_ptr points to the following structure:

```
dcl sru_number      fixed bin (32) aligned based;
```

For the following interrupt types:

suspend_ack
resume_ack
resume

info_ptr will be set to null, since there is no additional information.

For the following interrupt type:

establishment

info_ptr points to the following structure:

```
dcl 1 establishment_data_info aligned based,
      2 pc_data_ptr             ptr,
      2 status_info_ptr         ptr;
```

where:

pc_data_ptr
points to the pc_data structure as declared in the session_control_$ack_initiate entry.

status_info_ptr
points to the status_info structure as declared in the session_control_$status entry.

Entry: session_control_$recover

This entry allows the caller to recover the specified session.

Usage

```
dcl session_control_$recover entry (fixed bin (16), ptr,
                        fixed bin (35));

call session_control_$recover (session_id, recovery_info_ptr,
                        code);
```

where:

session_id (Input)
is the session identifier of the session to recover. This is returned by the initiate or accept_initiates entries.

recovery_info_ptr (Input)
is a pointer to the information necessary to recover a session. The structure pointed to is described below.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Notes

The structure used for recovery is as follows: (this declaration is available in session_dcls.incl.pl1)

```
dcl 1 recovery_info             aligned based,
    2 version                   fixed bin (35),
    2 sent_sru                  fixed bin (32),
    2 received_sru              fixed bin (32),
    2 reason                    fixed bin (16) unsigned,
    2 flags                     aligned,
     3 cancel                   bit (1) unaligned,
     3 my_turn                  bit (1) unaligned,
     3 preserve_interrupts      bit (1) unaligned,
     3 pad                      bit (33) unaligned;
```

where:

version
is the version of this structure. It should be set to recovery_info_v1.

sent_sru
is the sru number to be used for recovery of the sender's output data flow.

received_sru
is the sru number to be used for recovery of the sender's input data flow.

reason
is the reason for the recovery.

cancel
indicates whether or not records should be restarted from session control journals or queues.
"1"b restart from journals or queues
"0"b do not restart from journals or queues

my_turn
indicates whether the sender or the receiver should get the turn. Only meaningful on two way alternate sessions when cancel = "0"b.
"1"b sender gets the turn
"0"b receiver gets the turn

preserve_interrupts
is set to indicate whether or not the interrupt data flow of the session should be preserved or purged.
"1"b preserve the interrupt flow
"0"b purge the interrupt flow

pad
is reserved and must be zero.

Entry: session_control_$recover_ack

This entry allows the caller to acknowledge the recovery of the specified session.

Usage

```
dcl session_control_$recover_ack entry (fixed bin (16), ptr,
                        fixed bin (35));

call session_control_$recover_ack (session_id, recovery_info_ptr,
                        code);
```

where:

session_id (Input)
is the session identifier of the session on which the acknowledgement is to be sent. This is returned by the initiate or accept_initiates entries.

recovery_info_ptr (Input)
is a pointer to the information necessary to acknowledge the recovery of a session. The structure pointed to is described in the session_control_$recover entry. Notice that the reason and flags fields in the structure are ignored for an acknowledgement. The values of these fields are assumed to be the same as those sent in the recover request.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$resume

This entry allows the caller to resume a session which he has suspended using the session_control_$suspend entry. This call may only be made after the caller has received the "suspend_ack" interrupt.

Usage

```
dcl session_control_$resume entry (fixed bin (16), fixed bin (35));

call session_control_$resume (session_id, code);
```

where:

session_id (Input)
is the session identifier of the session to resume. This is returned by the initiate or accept_initiates entries.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$release_sru

This entry allows the caller to release a specified SRU, and any previous unreleased SRU on the specified session.

Usage

```
dcl session_control_$release_sru entry (fixed bin (16),
                        fixed bin (32), fixed bin (35));

call session_control_$release_sru (session_id,
                        sru_number, code);
```

where:

session_id (Input)
is the session identifier of the session on which to release the sru. This is returned by the initiate or accept_initiates entries.

sru_number (Input)
is the number of the sru to be released. Notice that this applies to the sender's input data flow.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$send

This entry sends data on a session with a specified enclosure and data type.

Usage

```
dcl session_control_$send entry (fixed bin (16), ptr,
                          fixed bin (21), ptr
                          fixed bin (21), fixed bin (35));

call session_control_$send (session_id, buf_ptr,
                          buf_len, record_info_ptr,
                          bytes_sent, code);
```

where:

session_id (Input)
is the session identifier on which data is to be sent. This is returned by the initiate or accept_initiates entries.

buf_ptr (Input)
is a pointer to a buffer which contains the data to be sent.

buf_len (Input)
is the length of the data buffer in 9-bit bytes.

record_info_ptr (Input)
is a pointer to a structure describing the record or data to be sent. The structure is defined under the session_control_$ack_initiate entry.

bytes_sent (Output)
is the number of bytes sent. If this is less than buf_len, the user will receive a wakeup on the data output event channel when more data can be accepted.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$status

This entry allows the caller to find out the status of the specified session.

Usage

```
dcl session_control_$status entry (fixed bin (16), ptr,
                        fixed bin (35));

call session_control_$status (session_id, status_info,
                        code);
```

where:

session_id (Input)
is the session identifier of the session for which status is desired. This is returned by the initiate or accept_initiates entries.

status_info (Input)
is a pointer to a structure in which the status is returned. The declaration for this structure is described below.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Notes

The structure filled in by this entry is as follows: (this declaration is available in session_dcls.incl.pl1)

```
dcl 1 status_info              aligned based,
      2 version                fixed bin (35),
      2 session_state          fixed bin,
      2 flags                  aligned,
        3 data_available       bit (1) unaligned,
        3 pad                  bit (35) unaligned,
      2 enclosure_available    fixed bin,
      2 session_info           aligned like session_type_descriptor;
```

where:

version
is the version of this structure. It should be set to status_info_v1.

session_state
is the current state of the session. The possible values and their meanings will be defined in MTB-532, "Internal Design of the DSA Session Control Implementation on Multics".

data_available
indicates whether or not data is available on the session.

enclosure_available
specifies the data enclosure level that is available. The values are described under session_control_$ack_initiate.

session_info
these are the values negotiated for all session characteristics. This structure is described under session_control_$accept_initates.

Entry: session_control_$suspend

This entry allows the caller to suspend a session. Normal data flow is not allowed on a suspended session, although, the session expedited data flow is not affected.

Usage

```
dcl session_control_$suspend entry (fixed bin (16),
                        fixed bin (16), fixed bin (35));

call session_control_$suspend (session_id, reason,
                        code);
```

where:

session_id (Input)
is the session identifier of the session which is to be suspended. This is returned by the initiate or accept_initiates entries.

reason (Input)
is the reason the session is being suspended.

code (Output)
is a standard system status code indicating the success or failure of the operation.

Entry: session_control_$terminate_abnormal

This entry allows the caller to send a terminate abnormal request on the specified session. After this request the session is terminated.

Usage

```
dcl session_control_$terminate_abnormal entry (fixed bin (16),
                        fixed bin (16) unsigned, fixed bin (35));

call session_control_$terminate_abnormal (session_id,
                        reason, code);
```

where:

session_id (Input)
is the session identifier of the session on which the request is to be sent. This is returned by the initiate or accept_initiates entries.

reason (Input)
is the reason code to be sent with the request.

code (Output)
is a standard system status code indicating the success or failure of the operation.

APPENDIX B

This appendix specifies the administrative gate interface which will be provided to session control. This interface will be used by the Network Administrator (NAD) as described in another MTB.

Name: session_control_admin_

This gate is the administrative interface to session control on Multics. The entries are described in the following sections. The entries available are:

session_control_admin_$initialize
session_control_admin_$add_route
session_control_admin_$delete_route
session_control_admin_$add_uncp
session_control_admin_$delete_uncp
session_control_admin_$add_mailbox
session_control_admin_$delete_mailbox
session_control_admin_$session_status
session_control_admin_$logical_connection_status
session_control_admin_$uncp_status
session_control_admin_$terminate_session
session_control_admin_$terminate_logical_connection
session_control_admin_$set_session_user

The calling sequences for these entries are not yet defined.