Stream,Function Name (Mnemonic)	*Direction*
S7,F31 Verification Request Send (VRS)	M,H->E,reply
Description	
This message requests the interpreting equipment to check the contents of the provided process program and inform the host whether or not the process program is acceptable for processing at the machine. The values of MDLN and SOFTREV are obtained from the PCD used to generate the process program. If S7,F31 is multi-block, it must be preceded by the S7,F1/S7,F2 Inquire/Grant transaction.	
Structure	

```
L,4
  1. <PPID>
  2. <MDLN>
  3. <SOFTREV>
  4. L,c                       (c = Number of Process Commands)
       1. L,2
            1. <CCODE>
            2. L,p             (p = Number of Parameters)
                 1. <PPARM1>
                 .
                 .
                 p. <PPARMp>
       2. L,2
       .
       .
       c. L,2
```

Exception
None

Stream,Function Name (Mnemonic)	*Direction*
S7,F32 Verification Request Acknowledge (VRA)	S,H<-E
Description	
Acknowledges reception of a formatted process program verification request at its destination and whether the process program was accepted by the equipment. A returned status of accepted by the interpreter means only that the message is understood. The validity of the contents of the process program is specified through a separate transaction (S7,F27/S7,F28).	
Structure	
`<ACKC7>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S7,F33 Process Program Available Request (PAR)	S,H<->E,reply
Description	
This message requests the interpreting host or equipment to check its process program library and tell the requester if the PPID will be supplied if requested.	
Structure	
`<PPID>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S7,F34 Process Program Availability Data (PAD)	S,H<->E
Description	
This message allows originator to tell requester whether it can provide the specified process program and whether it can provide it formatted, unformatted, or both.	
Structure	

`L,3`
`  1. <PPID>`
`  2. <UNFLEN>`
`  3. <FRMLEN>`	

Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S7,F35 Process Program for MID Request (PPMR)	S,H<->E,reply
Description	
This message is used to request the transfer of the process program to be used for the material identified.	
Structure	
`<MID>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S7,F36 Process Program for MID Data (PPMD)	M,H<->E
Description	
This message is used to transfer the process program for the material identified.	
Structure	

`L,3`
`  1. <MID>`
`  2. <PPID>`
`  3. <PPBODY>`	

Exception	
A zero-length list returned means no such MID or other error.	

Stream,Function Name (Mnemonic)	*Direction*
S7,F37 Large Process Program Send (LPPS)	S,H <-> E,reply
Description	
This is a request to send a process program via the Data Set Transfer protocol. The Data Set name, DSNAME, is the text string identifier of the process program, PPID. The Data Set is subsequently transferred as a Stream with the following internal SECSII structured data: `  <PPBODY>`	
Structure	
`<DSNAME>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S7,F38 Large Process Program Acknowledge(LPPA)	S,H <-> E
Description	
Acknowledge or error. A returned status of “accepted” means only that the message is understood. When the receiving entity is the equipment, there is aseparate verification transaction (For S7,F27/S7,F28) that indicates the completion status of the request. When the receiving entity is the host, the completion of the request is signaled by an event report.	
Structure	
`<ACKC7>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S7,F39 Large Formatted Process Program Send (LFPPS)	S,H <-> E, reply
Description	
This is a request to send a formatted process program via the Data Set Transfer Protocol. The Data Set name, DSNAME, is the text string identifier of the process program, PPID. The Data Set is subsequently transferred as a Stream with the following internal SECSII structured data:	

```
L,4
  1. <PPID>
  2. <MDLN>
  3. <SOFTREV>
  4. L,c                       (c = Number of Process Commands)
       1. L,2
            1. <CCODE>
            2. L,p              (p = Number of Parameters)
                 1. <PPARM1>
                 .
                 .
                 p. <PPARMp>
       2. L,2
       .
       .
       c. L,2
```

Structure	
`<DSNAME>`	
Exception	
none	

Stream,Function Name (Mnemonic)	*Direction*
S7,F40 Large Formatted Process Program Acknowledge (LFPPA)	S,H <-> E
Description	
Acknowledge or error. A returned status of “accepted” means only that the message is understood. When the receiving entity is the equipment, there is a separate verification transaction (S7,F27/S7,F28) that indicates the completion status of the request. When the receiving entity is the host, the completion of the request is signaled by an event report.	
Structure	
`<ACKC7>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S7,F41 Large Process Program Request(LPPR)	S,H <-> E, reply
Description	
This message is used to request the transfer of a process program via the Stream 13 Data set Transfer protocol. The Data Set name, DSNAME, is the text string identifier of the process program, PPID. The Data Set is subsequently transferred as a Stream with the following internal SECSII structured data: `<PPBODY>`	
Structure	
`<DSNAME>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S7,F42 Large Process Program Acknowledge (LPPA)	S,H <-> E
Description	
Acknowledge or error. A returned status of "accepted" means only that the message is understood. When the receiving entity is the equipment, there is a separate verification transaction (For example: S7,F27/S7,F28) that indicates the completion status of the request. When the receiving entity is the host, the completion of the request is signaled by an event report.	
Structure	
`<ACKC7>`	
Exception	
It is possible to use the ACKC7 code "command will be performed with completion signaled later" for this message.	

Stream,Function Name (Mnemonic)	*Direction*
S7,F43 Large Formatted Process Program Request (LFPPR)	S,H <-> E, reply
Description	
This message is used to request the transfer of a formatted process program via the Data set Transfer protocol. The Data Set name, DSNAME, is the text string identifier of the process program, PPID. The Data Set is subsequently transferred as a Stream with the following internal SECSII structured data:	

```
L,4
  1. <PPID>
  2. <MDLN>
  3. <SOFTREV>
  4. L,c                          (c = Number of Process Commands)
       1. L,2
            1. <CCODE>
            2. L,p               (p = Number of Parameters)
                 2. <PPARM1>
                 .
                 .
                 p. <PPARMp>
       2. L,2
       .
       .
       c. L,2
```

Structure
`<DSNAME>`
Exception
None

Stream,Function Name (Mnemonic)	*Direction*
S7,F44 Large Formatted Process Program Acknowledge (LFPPA)	S,H <-> E
Description	
Acknowledge or error. A returned status of "accepted" means only that the message is understood. When the receiving entity is the equipment, there is a separate verification transaction (S7,F27/S7,F28) that indicates the completion status of the request. When the receiving entity is the host, the completion of the request is signaled by an event report.	
Structure	
`<ACKC7>`	
Exception	
It is possible to use the ACKC7 code "command will be performed with completion signaled later" for this message.	

10.12 *Stream 8 Control Program Transfer* — The purpose of this stream is to provide the method for transmitting the programs used in the equipment to perform the control function or to execute the transmitted process program.

Stream,Function Name (Mnemonic)	*Direction*
S8,F0 Abort Transaction (S8F0)	S,H<->E
Description	
Same form as S1,F0.	
Structure	
Exception	

Stream,Function Name (Mnemonic)	*Direction*
S8,F1 Boot Program Request (BPR)	S,H<->E,reply
Description	
This message is used to request the transmission of the boot program. It is assumed that there is only one boot program associated with any given equipment.	
Structure	
`Header only`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S8,F2 Boot Program Data (BPD)	M,H<->E
Description	
The boot program is required by some systems as a precursor to loading an operating system or executive program.	
Structure	
`<BPD>`	
Exception	
A zero-length item means no boot.	

Stream,Function Name (Mnemonic)	*Direction*
S8,F3 Executive Program Request (EPR)	S,H<->E,reply
Description	
This message is used to request the executive program. It is assumed that there is only one executive program associated with any given equipment.	
Structure	
`Header only`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S8,F4 Executive Program Data (EPD)	M,H<->E
Description	
The executive program is the master control program of the equipment. The executive may contain all the program required or it may contain the information required to request the rest of the program it needs on Stream 13.	
Structure	
`<EPD>`	
Exception	
None	

10.13 *Stream 9 System Errors* — This stream provides a method of informing the host that a message block has been received which cannot be handled or that a timeout on a transaction (receive) timer has occurred. The messages indicate either a Message Fault or a Communications Fault has occurred but do not indicate a Communications Failure has occurred.

10.13.1 *Communications Failure* — A Communications Failure occurs in a SECS-I environment when, and only when, the RTY limit is exceeded.

NOTE 10: In the event of a Communications Failure, no Stream 9 message is sent.

10.13.2 *Communications Fault* — A Communications Fault occurs when the equipment does not receive an expected message (when a transaction timer or a conversation timer has expired).

10.13.3 *Message Fault* — A Message Fault occurs when the equipment receives a message which it cannot process because of a fault that arises from the content, context, or length of the message.

Stream,Function Name (Mnemonic)	*Direction*
S9,F0 Abort Transaction (S9F0)	S,H<->E
Description	
Same form as S1,F0.	
Structure	
Exception	

Stream,Function Name (Mnemonic)	*Direction*
S9,F1 Unrecognized Device ID (UDN)	S,H<-E
Description	
The device ID in the message block header did not correspond to any known device ID in the node detecting the error.	
Structure	
`<MHEAD>`	
Exception	
None	

S9,F2 Not Used

Stream,Function Name (Mnemonic)	*Direction*
S9,F3 Unrecognized Stream Type (USN)	S,H<-E
Description	
The equipment does not recognize the stream type in the message block header.	
Structure	
`<MHEAD>`	
Exception	
None	

S9,F4 Not Used

Stream,Function Name (Mnemonic)	*Direction*
S9,F5 Unrecognized Function Type (UFN)	S,H<-E
Description	
This message indicates that the function in the message ID is not recognized by the receiver.	
Structure	
`<MHEAD>`	
Exception	
None	

S9,F6 Not Used

Stream,Function Name (Mnemonic)	*Direction*
S9,F7 Illegal Data (IDN)	S,H<-E
Description	
This message indicates that the stream and function were recognized, but the associated data format could not be interpreted.	
Structure	
`<MHEAD>`	
Exception	
None	

S9,F8 Not Used

Stream,Function Name (Mnemonic)	*Direction*
S9,F9 Transaction Timer Timeout (TTN)	S,H<-E
Description	
This message indicates that a transaction (receive) timer has timed out and that the corresponding transaction has been aborted. It is up to the host to respond to this error in an appropriate manner to keep the system operational.	
Structure	
`<SHEAD>`	
Exception	
None	

S9,F10 Not Used

Stream,Function Name (Mnemonic)	*Direction*
S9,F11 Data Too Long (DLN)	S,H<-E
Description	
This message to the host indicates that the equipment has been sent more data than it can handle.	
Structure	
`<MHEAD>`	
Exception	
None	

S9,F12 Not Used

Stream,Function Name (Mnemonic)	*Direction*
S9,F13 Conversation Timeout (CTN)	S,H<-E
Description	
Data were expected but none were received within a reasonable length of time. Resources have been cleared.	
Structure	

`L,2`
`  1. <MEXP>`
`  2. <EDID>`	

Exception	
None	

S9,F14 Not Used

10.14 *Stream 10 Terminal Services* — The functions of this stream is to pass textual messages between operator terminals attached to processing and/or testing equipment and the host. The equipment makes no attempt to interpret the text of the message, but merely passes it from terminal keyboard to the host or from the host to the display of the terminal. Management of human response times to information displayed on terminals is the responsibility of the host.

Stream,Function Name (Mnemonic)	*Direction*
S10,F0 Abort Transaction (S10F0)	S,H<->E
Description	
Same form as S1,F0.	
Structure	
Exception	

Stream,Function Name (Mnemonic)	*Direction*
S10,F1 Terminal Request (TRN)	S,H<-E,[reply]
Description	
A terminal text message to the host.	
Structure	

`L,2`
`  1. <TID>`
`  2. <TEXT>`	

Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S10,F2 Terminal Request Acknowledge (TRA)	S,H->E
Description	
Acknowledge or error	
Structure	
`<ACKC10>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S10,F3 Terminal Display, Single (VTN)	S,H->E, [reply]
Description	
Data to be displayed.	
Structure	

`L,2`
`  1. <TID>`
`  2. <TEXT>`	

Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S10,F4 Terminal Display, Single Acknowledge (VTA)	S,H<-E
Description	
Acknowledge or error	
Structure	
`<ACKC10>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S10,F5 Terminal Display, Multi-Block (VTN)	M,H->E,[reply]
Description	
Data to be displayed on the equipment's terminal.	
Structure	
L,2 1. <TID> 2. L,n 1. <TEXT$_1$> . . n.<TEXT$_n$>	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S10,F6 Terminal Display, Multi-block Acknowledge (VMA)	S,H<-E
Description	
Acknowledge or error	
Structure	
`<ACKC10>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S10,F7 Multi-block Not Allowed (MNN)	S,H<-E
Description	
An error message from a terminal that cannot handle a multi-block message from S10,F5.	
Structure	
`<TID>`	
Exception	
None	

S10,F8 Not Used

Stream,Function Name (Mnemonic)	*Direction*
S10,F9 Broadcast (BCN)	S,H->E,[reply]
Description	
This function is generally the same as S10,F3 except that specific TID in each equipment need not be specified. Instead, the text is directed to each terminal in the equipment when the function is received. This function assumes that this feature exists on all equipment, otherwise repeated S10,F3 messages should be used.	
Structure	
`<TEXT>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S10,F10 Broadcast Acknowledge (BCA)	S,H<-E
Description	
Acknowledge or error	
Structure	
`<ACKC10>`	
Exception	
None	

10.15 Stream 11 has been deleted and will not appear again in this publication.

10.15.1 It is the consensus of the Communications Committee that Stream 11 is obsolete. Its use is discouraged, and it has been removed from the 1989 edition of the standard. The reasons for removal are three-fold:

1. The purpose of this stream, as it was originally envisioned, is perceived to be of little use and can best be accomplished by other means beyond the scope of this standard;
2. The functions in this stream have many technical problems that severely limit their use;
3. There is a noticeable lack of implementations of this standard that utilize Stream 11 in its originally intended form.

NOTE 11: Applications that need to transfer unformatted data between the host and equipment should use the facilities of Stream 13.

10.16 *Stream 12 Wafer Mapping* — Messages which deal with coordinate positions and data associated with those positions. This includes functions such as wafer mapping with coordinates of die on a wafer and the associated binning information.

10.16.1 *Structure* — Functions 1 through 20 address the variations required by semiconductor equipment manufacturers in transmitting wafer maps to and from the process equipment (wafer probe through die attach). The functions include three basic formats. The three formats developed are:

1. Row/column format where a coordinate row starting position is given with die count in the row and starting direction. The respective binning information follows for each die.
2. Array format is structured such that a matrix array captures all or part of a wafer with the associated binning information.
3. Coordinate format provides an X/Y location and bin code for die on the wafer.

10.16.2 *Definitions and Descriptions* — The following information is required to perform map association to the physical wafer as it relates to the archival use and transmission of wafer maps.

1. Flat/Notch Location
2. Frame Rotation
3. Row Count
4. Column Count
5. Die Units of Measure
6. Die Size
7. Process Die Count
8. Reference Points
9. Bin Code Equivalents
10. Process Axis
11. Null Bin Code Value
12. ID Type

10.16.2.1 *Flat/Notch Location* — The position in degrees that the flat or notch are oriented during processing relative to a "normal" position of zero degrees. See Figure 6.

10.16.2.2 *Frame Rotation* — The orientation of a film frame relative to a "normal" position of zero degrees. See Figure 7.

10.16.2.3 *Row/Column Count* — The row and column counts are the total number of rows and columns, respectively, on a wafer which must be correlated directly with the wafer map. These numbers will always be greater than zero.

10.16.2.4 *Die Sizes* — The die size is given in standard units as specified by the die unit of measure item DUTMS, and will also be greater than zero. The value of the die size is determined by measuring the distance from a point on one die to the same point on the next die, often referred to as an index. This is depicted in the lower portion of Figure 7, Section B in the General Rules Section.

10.16.2.5 *Process Die Count* — The process die count item is used by equipment that is being map driven to make determinations about how much material to prepare. For example, a die attach will epoxy lead frames in advance of the attach process. By knowing the total number of die to be processed within a wafer map, the equipment can stop epoxying lead frames equivalent to the last die to be attached. This item is also used by the equipment to tell the host how many total die it processed for that map. For example, a die attach would use PRDCT to report the total die actually attached from a particular wafer.

10.16.2.6 *Reference Points* — Reference points provide a means of relating a map to the physical wafer. The total number of these points, and the method for assigning and detecting them, is the responsibility of the equipment. This standard only provides a means for transmitting them.

10.16.2.7 *Origin* — The origin is in one of five locations which is specified by the equipment when generating a wafer map. The origin is on an array structure having dimensional values equal to those specified by the row and column count. The origin then lies on one of the four corners of that array or in a center location determined by the following formula:

$$\left(\frac{\text{row} \sim \text{or} \sim \text{column} + 1}{2}\right)\text{truncated}$$

10.16.2.7.1 It is implicit in determining the center location that the upper-left-hand corner of the area, in the normal position, be counted as the first row and column position. An equipment requesting a map provides the origin location that it wants the map to be based on before transmission. If the equipment does not provide an origin, the host must provide a default value. An equipment transmitting a map must provide the origin with the map setup data.

10.16.2.8 *Bin Code Equivalents* — Bin code equivalents is a list of bin codes that the receiving equipment will process. (i.e., if a map contains codes 1 through 10 and the good die are bins 1 and 2, then bin code equivalent list could indicate 1 and 2 if only the good die categories were needed. These are the only bin codes to which an equivalent will drive for its respective process function.) In the case of X/Y coordinate format, the locations transmitted will be only those with the bin codes stated in the Bin Code Equivalent list, unless the length byte is set to zero, in which all of the bin codes in the map will be transmitted.

10.16.2.9 *Process Axis* — The process axis is the axis, either rows or columns, increasing or decreasing, and the side of the map, (top, bottom, left, or right, respectively) that the map data will originate from. This is based on the coordinate system as described under the General Rules section of this document.

10.16.2.10 *ID Type* — ID type indicates the appropriate material ID type (i.e., wafer, cassette, or film frame).

10.16.3 *General Rules*

10.16.3.1 *Map Data Size* — Stream 12 provides for the transmission of a complete map regardless of size. Equipment requiring segmented maps for transmission or reception will not be able to use the Stream 12 functions to handle the complete conversation.

10.16.3.2 *Orientation Conventions* — The orientation of a wafer presented for processing will differ from equipment to equipment. Stream 12 specifies conventions for expressing wafer orientation so that a map can be translated from one geometric representation to another.

10.16.3.2.1 The bottom of the wafer is the notch or the line of the major flat. The orientation of a wafer is measured in positive degrees clockwise (CW) from the "normal" position. The "normal" position is where the bottom of the wafer is closest to you when the wafer is lying horizontally in front of you with the die side facing up. The "normal" position has an orientation of zero degrees. See Figure 6 for graphic representation of wafer orientations.

10.16.3.2.2 The bottom of a film frame is also the notch or the line of notches. Its orientation and "normal" position are measured in the same manner as for wafers. See Figure 7 for examples of bottoms of film frames.

10.16.3.2.3 The orientation of an unmounted wafer presented for processing is given by the parameter FNLOC, Flat/Notch LOCation.

10.16.3.2.4 The ultimate orientation of a wafer presented for processing after it has been mounted on a film frame is the cumulative rotation of the wafer from the "normal" position on the film frame and the rotation of film frame as it is presented to the equipment. This is determined by the sum of the parameters FNLOC and FFROT, Film Frame ROTation. It is possible for an application to represent the ultimate orientation of a wafer in one of these parameters only and pass the other parameter as zero length.

10.16.3.2.5 Figure 6 shows wafers oriented at 270 degrees with respect to the bottoms of a metal and round film frame. If one of these film frames were presented to an equipment rotated 90 degrees clockwise (CW), (bottom facing the left edge of the page), the ultimate orientation of the wafer would be zero degrees.

10.16.3.2.6 In the case where either FNLOC or FFROT are unknown or irrelevant information, a zero-length data item is transmitted, and the item will be ignored by the application. One of the items must exist.

10.16.3.3 *Coordinate Axis System* — The coordinate axis orientation is shown in Figure 8, Section A. The assumption is that the "X" or "column" coordinates increase to the right of the "Y-axis" and the "Y" or "row" coordinates increase above the "X-axis." In describing the physical wafer it is also given that the coordinate axis orientation never rotates. The wafer moves or rotates within the coordinate axis system. The origin within the array describing the wafer's coordinate system must be in one of five locations on that array (the center, upper-left, lower-left, upper-right, or lower-right corner of the array).

10.16.3.3.1 Figures 9 and 10 summarize the conversation protocol in the form of a flow chart. Since a single transmit inquire/grant can be used for one of three message function pairs, the application is required to examine MAPFT, is received as part of the map setup data to determine the appropriate function to follow. If the appropriate function is not transmitted, the conversation is aborted and the error is reported using the appropriate error reporting stream and function.

Stream,Function Name (Mnemonic)	*Direction*
S12,F0 Abort Transaction (S12F0)	S,H<->E
Description	
Same form as S1F0.	
Structure	
Exception	

Stream,Function Name (Mnemonic)	*Direction*
S12,F1 Map Set-up Data Send (MSDS)	S,H<-E,reply
Description	
Used to send all of the map set-up data common to all formats and required to link the data map with the physical wafer.	
Structure	
L,15 1. <MID> 2. <IDTYP> 3. <FNLOC> 4. <FFROT> 5. <ORLOC> 6. <RPSEL> 7. L,n 1. <$REFP_xREFP_y$> . . n. <$REFP_xREFP_y$> 8. <DUTMS> 9. <XDIES> 10. <YDIES> 11. <ROWCT> 12. <COLCT> 13. <NULBC> 14. <PRDCT> 15. <PRAXI>	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S12,F2 Map Set-up Data Acknowledge (MSDA)	S,H->E
Description	
Acknowledgment of receipt of complete set of map set-up parameters.	
Structure	
`<SDACK>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S12,F3 Map Set-up Data Request (MSDR)	S,H<-E,reply
Description	
Used to request set-up data from the host for the product ready to be processed at the equipment (common to all formats).	
Structure	

`L,9`
`  1. <MID>`
`  2. <IDTYP>`
`  3. <MAPFT>`
`  4. <FNLOC>`
`  5. <FFROT>`
`  6. <ORLOC>`
`  7. <PRAXI>`
`  8. <BCEQU...>`
`  9. <NULBC>`	

Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S12,F4 Map Set-up Data (MSD)	S,H->E
Description	
Used to send all of the map set-up data required to link the data map with the physical wafer.	
Structure	

`L,15`
`   1. <MID>`
`   2. <IDTYP>`
`   3. <FNLOC>`
`   4. <ORLOC>`
`   5. <RPSEL>`
`   6. L,n`
`        1. <REFPxREFPy>`
`        .`
`        .`
`        n. <REFPxREFPy>`
`   7. <DUTMS>`
`   8. <XDIES>`
`   9. <YDIES>`
`  10. <ROWCT>`
`  11. <COLCT>`
`  12. <PRDCT>`
`  13. <BCEQU>`
`  14. <NULBC>`
`  15. <MLCL>`	

Exception	
A zero-length list returned means no such MID.	

Stream,Function Name (Mnemonic)	*Direction*
S12,F5 Map Transmit Inquire (MAPTI)	S,H<-E,reply
Description	
Used to prepare the host for map transmission. S12,F5 must precede all S12,F7-8,F9-10, & F11-12 transactions.	
Structure	

`L,4`
`  1. <MID>`
`  2. <IDTYP>`
`  3. <MAPFT>`
`  4. <MLCL>`	

Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S12,F6 Map Transmit Grant (MAPTG)	S,H->E
Description	
Provides permission to transfer.	
Structure	
`<GRNT1>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S12,F7 Map Data Send Type 1 (MDS1)	M,H<-E,reply
Description	
Used to send map data from the equipment to the host in row or column compressed format. If S12,F7 is multi-block, it must be preceded by the S12,F5/S12,F6 Inquire/Grant transaction.	
Structure	

```
L,3
  1. <MID>
  2. <IDTYP>
  3. L,n
       1. L,2
            1. <RSINF1>
            2. <BINLT1>
       2. L,2
       .
       .
       n. L,2
            1. <RSINFn>
            2. <BINLTn>
```

Exception
None

Stream,Function Name (Mnemonic)	*Direction*
S12,F8 Map Data Acknowledge Type 1 (MDA1)	S,H->E
Description	
Acknowledge or error	
Structure	
`<MDACK>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S12,F9 Map Data Send Type 2 (MDS2)	M,H<-E,reply
Description	
Used to send map data from the equipment in array format. If S12,F9 is multi-block, it must be preceded by the S12,F5/S12,F6 Inquire/Grant transaction.	
Structure	

```
L,4
  1. <MID>
  2. <IDTYP>
  3. <STRPxSTRPy>
  4. <BINLT...>
```

Exception
None

Stream,Function Name (Mnemonic)	*Direction*
S12,F10 Map Data Acknowledge Type 2 (MDA2)	S,H->E
Description	
Acknowledge or error	
Structure	
`<MDACK>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S12,F11 Map Data Send Type 3 (MDS3)	M,H<-E,reply
Description	
Used to send map data from the equipment in cartesian coordinate format. Bin values may or may not be included in the message. If S12,F11 is multi-block, it must be preceded by the S12,F5/S12,F6 Inquire/Grant transaction.	
Structure	

```
L,3
  1. <MID>
  2. <IDTYP>
  3. L,n
       1. L,2
            1. <XYPOS1x XYPOS1y>
            2. <BINLT1...>
       2. L,2
       .
       .
       n. L,2
            1. <XYPOSn>
            2. <BINLTn>
```

Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S12,F12 Map Data Acknowledge Type 3 (MDA3)	S,H->E
Description	
Acknowledge or error	
Structure	
`<MDACK>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S12,F13 Map Data Request Type 1 (MDR1)	S,H<-E,reply

Description

Used to request map data for product at equipment process station in row or column format.

Structure

```
L,2
  1. <MID>
  2. <IDTYP>
```

Exception

None

Stream,Function Name (Mnemonic)	*Direction*
S12,F14 Map Data Type 1 (MD1)	M,H->E

Description

Used to send map data from the host to the equipment in row or column format.

Structure

```
L,3
  1. <MID>
  2. <IDTYP>
  3. L,n
       1. L,2
            1. <RSINF$_{x1}$ RSINF$_{y1}$RSIN$_{d}$>
            2. <BINLT...>
       2. L,2
       .
       .
       n. L,2
            1. <RSINF$_{n}$>
            2. <BINLT$_{n}$>
```

Exception

None

Stream,Function Name (Mnemonic)	*Direction*
S12,F15 Map Data Request Type 2 (MDR2)	S,H<-E,reply

Description

Used to request map data for product at an equipment process station, in array format.

Structure

```
L,2
  1. <MID>
  2. <IDTYP>
```

Exception

None

Stream,Function Name (Mnemonic)	*Direction*
S12,F16 Map Data Type 2 (MD2)	M,H->E

Description

Used to send map data from the host to the equipment in array format.

Structure

```
L,4
  1. <MID>
  2. <IDTYP>
  3. <STRP_x STRP_y>
  4. <BINLT...>
```

Exception

A zero-length list returned means no such MID.

Stream,Function Name (Mnemonic)	*Direction*
S12,F17 Map Data Request Type 3 (MDR3)	S,H<-E,reply

Description

Used to request map data for product at an equipment process station in cartesian coordinate format.

Structure

```
L,3
  1. <MID>
  2. <IDTYP>
  3. <SDBIN>
```

Exception

None

Stream,Function Name (Mnemonic)	*Direction*
S12,F18 Map Data Type 3 (MD3)	M,H->E

Description

Used to send map data from the host to the equipment in cartesian coordinate format. Bin values may or may not be included.

Structure

```
L,3
  1. <MID>
  2. <IDTYP>
  3. L,n
       1. L,2
            1. <XYPOS_x1 XYPOS_y1>
            2. <BINLT_1...>
       2. L,2
       .
       .
       n. L,2
            1. <XYPOS_n>
            2. <BINLT_n>
```

Exception

A zero-length list returned means no such MID.

Stream,Function Name (Mnemonic)	*Direction*
S12,F19 Map Error Report Send (MERS)	S,H<->E
Description	
Used to transmit map related errors.	
Structure	

`L,2`
`  1. <MAPER>`
`  2. <DATLC>`	

Exception	
None	

S12,F20 Not Used

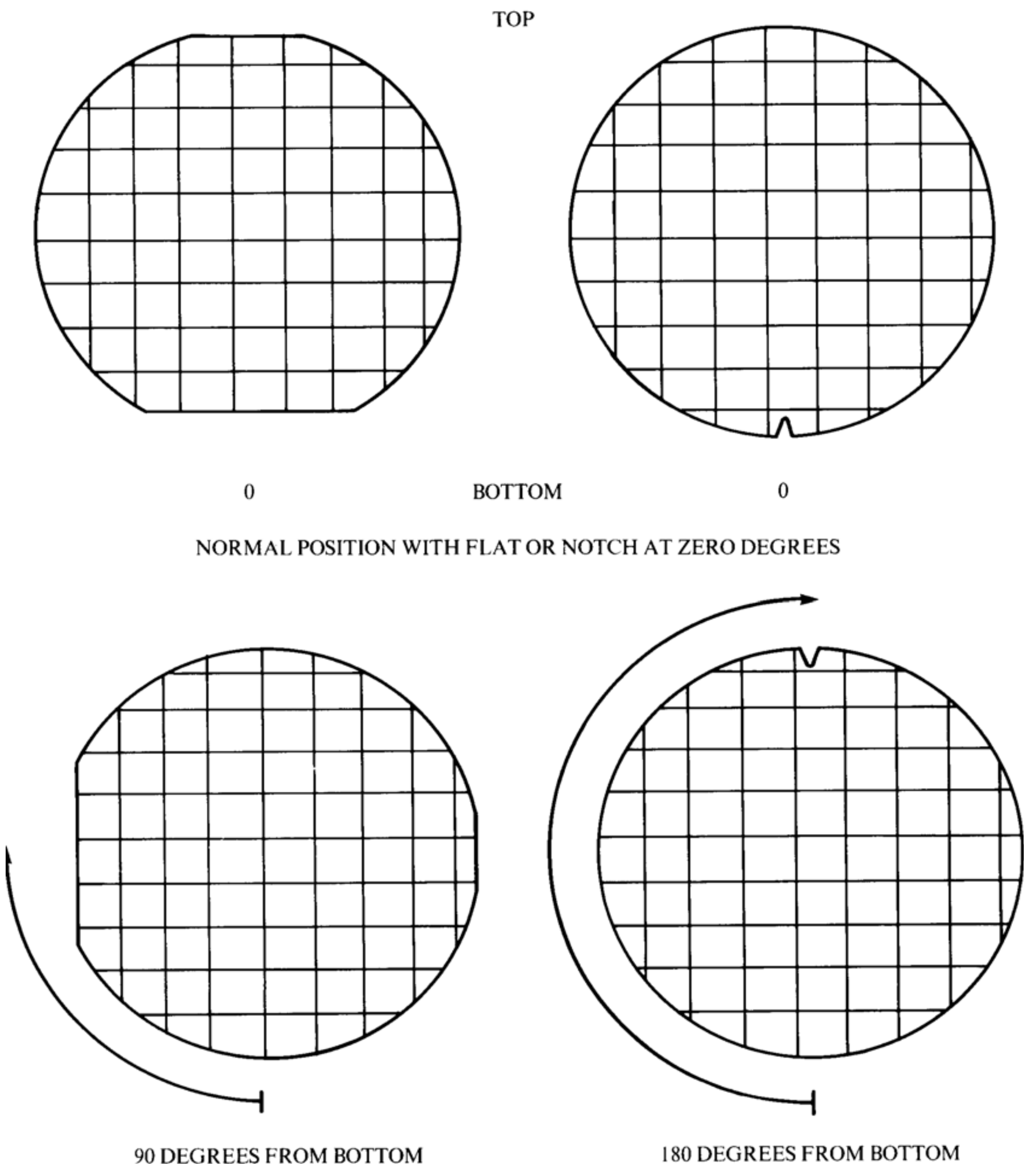

Figure 6
Wafer Rotation Position in Degrees

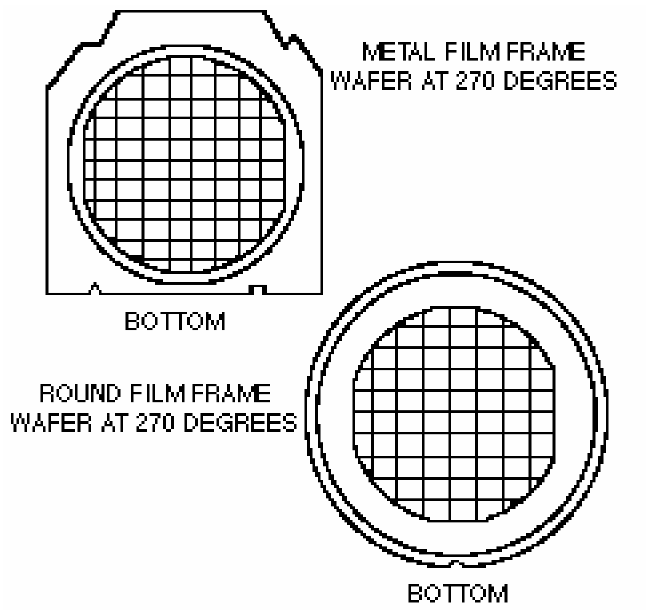

Figure 7
Wafer Rotation on Film Frame

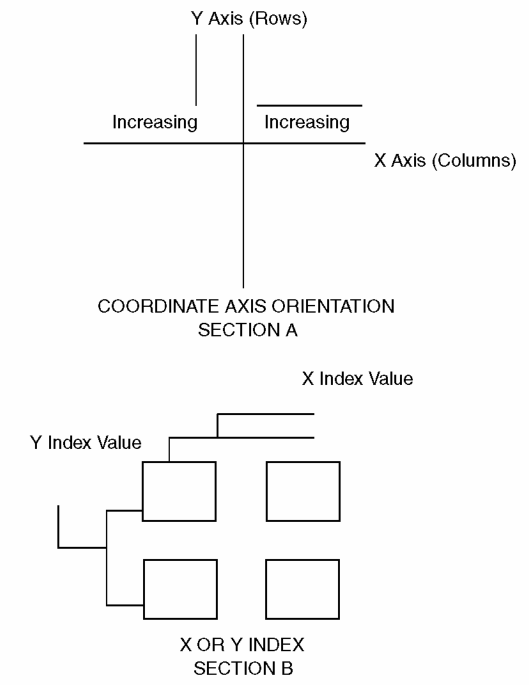

Figure 8
Orientation Reference and Index Determination

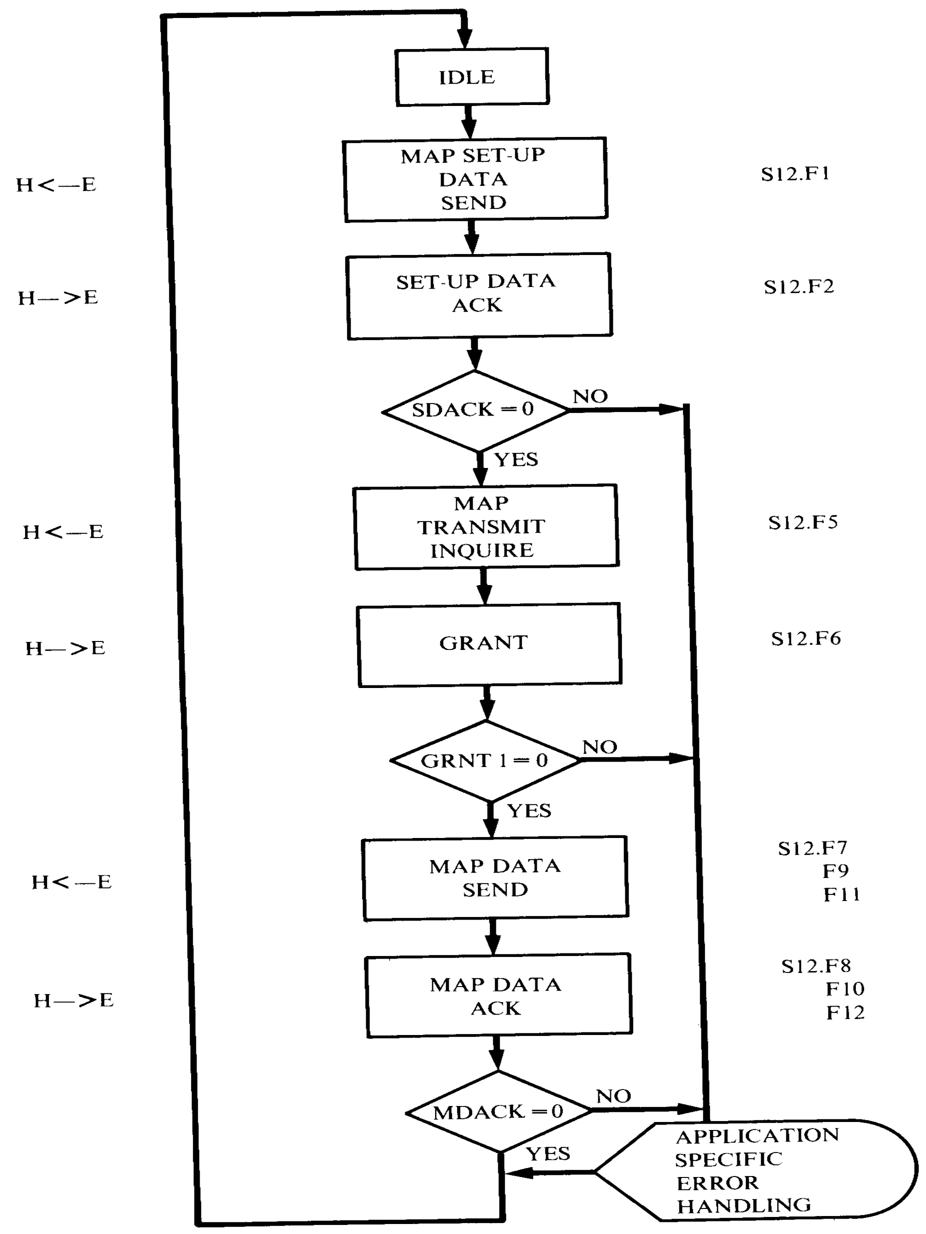

Figure 9
Wafer Map Transmitted by Equipment

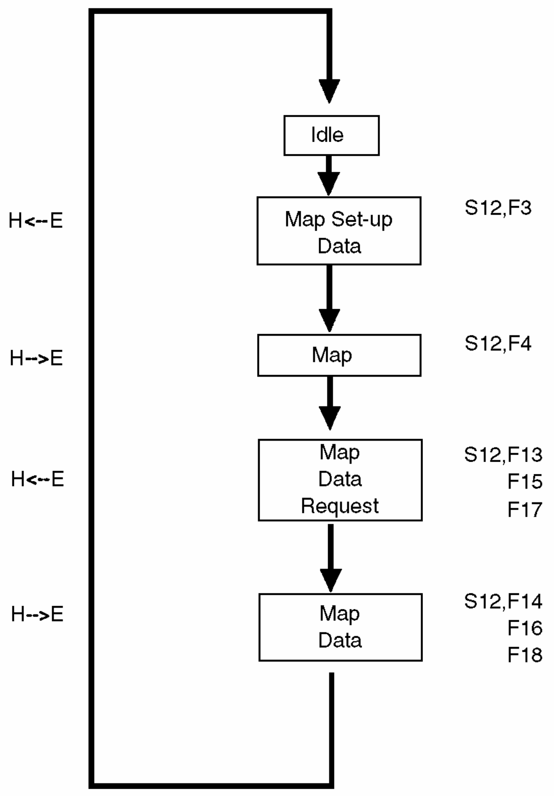

Figure 10
Wafer Map Received by Equipment

10.17 *Stream 13 Data Set Transfers* — This stream provides protocols to transfer data sets between systems. It is not intended to provide a general file access mechanism.

10.17.1 *Data Set Characteristics* — The data set may reside on the host or the equipment.

10.17.1.1 The term *data set* is used in a very general sense. A data set may represent a file, a data structure in memory, a collection of sensor values, or high density wafer profile data. The protocols define only the way data is sent from one system to another and do not define how the data set is stored by either the host or equipment.

10.17.1.2 The *sending* system is defined to be the system that has the data set. The *receiving* system is defined to be the system to which the data set is being transferred. The host or the equipment may assume either role.

10.17.2 *Unformatted Data Set Protocol* — The protocol for transferring unformatted data sets has the following characteristics:

1. Information about the record structure of the data set may be available.
2. ASCII records are transferred without the record terminating “noise” characters used by some operating systems.
3. Data sets do not need to be transferred in a single message.
4. No arbitrary limits are imposed on the length of one message. The maximum amount of data sent in each message is determined by both the sending and receiving systems, so there can be no data overruns.
5. There is a method of restarting a transfer in the event of an interruption.

10.17.2.1 *Data Set Name* — An ASCII string (format 20) which performs a function similar to the Message ID (Stream and Function) of the SECS-I protocol. This is a logical name which has meaning to both the equipment and the host. Neither the equipment nor the host is required to use the *Data Set Name* for the same information in any other context. For example, maintenance data may be stored in the equipment in the file "WIDGET.DAT" and in the host as records in a database, but the *Data Set Name* may be "S11,F2".

10.17.2.2 *Records* — The Record Type determines the way the data set is divided into messages for transfer to the receiving system. There are two types of records: Discrete and Stream.

1. A data set with *Discrete* records has a traditional record structure, such as ASCII text. *RecordLength* is the length of the longest record. Zero-length records are allowed. Each record from the data set is sent as a single item in a message.
2. If *Record Type* is *Stream,* then the data set has no internal structure which can be communicated with this protocol.

These kind of data set might be, for example, a dump of main memory, SECS-II structured data, or data which has implicit record boundaries. RecordLength has no meaning for this kind of data set. Items containing data from the data set have no relationship to the structure of the data set.

10.17.2.3 *Transactions* — The basic data transfer is performed by the OPEN, READ, and CLOSE transactions. There is no explicit write transaction. A write is performed indirectly using the SEND transaction. A RESET transaction is provided to allow a graceful recovery after a crash. This protocol describes the transactions over the communications channel only. No assumption is made about the implementation of the transactions. For example, the OPEN transaction on a data set which is stored on a disk file does not necessarily cause the sending system to open the file.

10.17.2.3.1 The OPEN, READ, and CLOSE transactions are initiated by the receiving system. The SEND transaction is initiated by the sending system. The RESET transaction is initiated by either system. The usual transaction timer operates between the primary and secondary messages of each transaction. The time between transactions, especially between READ transactions, is application-specific so no additional timer is defined.

10.17.2.3.2 Internally, the protocol uses a *Handle* to keep track of multiple open data sets, and a *Checkpoint* which aids in error recovery. A value called *ReadLength* is used to negotiate the amount of data sent at one time.

10.17.2.4 *Handle* — Between the sending and receiving systems, more than one data set may be open at a time, or one data set may be opened many times. The Handle is a short name used to keep track of the state of a particular data set and instance of OPEN between the sending and receiving systems (see Figure 11). This *Handle* may be thought of as a name for a single application level connection from the sending to the receiving system. Its value is assigned in the primary message of the OPEN transaction.

10.17.2.4.1 The value used for the *Handle* must not be used in another OPEN by the receiving system to the same sending system until it is used in a CLOSE to that sending system, or the RESET transaction is sent by either system. For example, assume a host system opens a data set on equipment 255 using *Handle 1.* The host may not issue another OPEN to equipment 255 using *Handle 1* until it closes 1 on 255. However, the host may use *Handle 1* to open a data set on another piece of equipment, and equipment 255 may use *Handle 1* to open a data set on the host.

10.17.2.4.2 The number of data sets which may be open at one time and the number of times one data set may be opened is not specified by this standard. Error codes are defined for situations where the limits are exceeded. It must be possible to have one outstanding transaction (i.e., a primary message for which there has not been a reply) for each open *Handle*. If the sending system receives a primary message for a Handle which already has an outstanding transaction, then the error code for Pending Transaction is returned in the secondary message (see Figure 11).

10.17.2.5 *Checkpoint* — The response to each READ transaction contains the data and a new Checkpoint value. The *Checkpoint* is defined by and has meaning only for the sending system. Its purpose is to allow a data transfer to be restarted from the point of the last complete message after some communication interruption. The exact nature of the *Checkpoint* is not specified. It could be the byte index in the data set, a record counter (for Discrete records), or some other system-dependent value.

NOTE 12: Checkpoint and the SECS-II transaction timer define a performance requirement for the sending system. The sending system must be able to get data from any checkpoint location within a data set between the receipt of the OPEN primary message and the time for reply to the first READ.

10.17.2.5.1 The value of *Checkpoint* must conform to several rules:

1. The Checkpoint value is exactly four bytes long.

2. The beginning of the data set has Checkpoint value with all bits reset.
3. A Checkpoint with all bits set is illegal.
4. A Checkpoint supplied by the sending system which does not have all bits set is usable in an OPEN transaction to restart a data transfer without any lost data or duplicated data.

10.17.2.5.2 The receiving system defines the initial *Checkpoint* in the primary message of the OPEN transaction. The sending system returns the next *Checkpoint* in the response to each READ.

10.17.2.6 *Read Length* — The *Read Length* must be supplied by the receiving system with each READ transaction. It specifies the maximum number of data bytes which that system is prepared to process at one time. The sending system may supply less if it has limited resources. The sending system may supply more if *ReadLength* is zero, or is smaller than *RecordLength*.

10.17.2.7 *Reading a Data Set* — The basic data transfer is initiated with the OPEN transaction and completed with the CLOSE transaction. Information is sent from the sending to the receiving systems by a series of READ transactions.

10.17.2.8 *OPEN Transaction* — The receiving system sends a primary message containing the *DataSet Name* of the desired data set, the *Handle* to be used, and the *Checkpoint* of the initial READ transaction. The response from the sending system is a secondary message with a return code and the *RecordType* and *RecordLength* of the data set. If the return code is one of the error codes, then no data set was opened and the values of *RecordType* and *RecordLength* are undefined. If the *RecordType* is *Stream*, then the value of *RecordLength* is undefined. Notice that the undefined items will still appear in the secondary message.

10.17.2.8.1 The return code in the secondary message is one of the following:

OK.

ERROR: Unknown Data Set ID.

ERROR: Try later (i.e., the data set is in use).

ERROR: Too many open data sets.

ERROR: Data set open too many times.

ERROR: Handle in use.

ERROR: Pending Transaction.

10.17.2.9 *The READ Transaction* — The receiving system sends a primary message which contains the *Handle*, and the *ReadLength*. The sending system responds with a secondary message which has a return code, the next Checkpoint, and zero or more items with data. At least one data item must be supplied unless there is an error. The return codes are:

OK.

ERROR: End of Data.

ERROR: No open Data Set (i.e., incorrect Handle).

ERROR: Cannot continue (i.e., a disk read error on the sending system).

ERROR: Pending Transaction.

10.17.2.9.1 Any READ transaction which follows a READ which returned an error, except "Pending Transaction," will generate the same error. The value of *Checkpoint* must be illegal (i.e., all bits must be set) when the "End of Data," "No open Data Set," or "Pending Transaction" error is returned. The value of *Checkpoint* error must be a value from which recovery may be attempted without duplicating data when the "Cannot continue" error is returned. Recovery may be attempted by issuing a CLOSE, followed by an OPEN with the last value of *Checkpoint*, and then another READ.

10.17.2.9.2 Each secondary message for the READ transaction must contain a whole number of *Discrete* records. A record may be sent as an ASCII or a binary item. *Stream* data sets are broken into pieces by the READ transaction without regard to internal structure. Each piece would be sent as a single binary item. The number of items which contain data depends on the *RecordType, Record Length,* and *Read Length*. The algorithm is designed so that the maximum length of the secondary message is deterministic. It gives the receiving system the ability to control the amount of resources (such as SECS-I buffers) which it must allocate. The sending system may send less data than the maximum if it has limited resources. The performance (i.e., the packing of records into a message of some maximum size) should be very good for the case where records are all nearly the maximum length. This is assumed to be the usual case. The efficiency in pathological cases (e.g., many short records) will not be good, but the algorithm is robust enough to accommodate this without exceeding the maximum message size.

NOTE 13: If the *RecordType* is *Stream,* then there is exactly one item with a binary format whose length is not more than *ReadLength*. If the *RecordType* is *Discrete,* then the maximum number of items, MaxItems, is calculated by the formula:

$$\text{MaxItems} = \max\left(1, \text{int}\left(\frac{\text{ReadLength}}{\text{RecordLength}}\right)\right)$$

10.17.2.9.3 The size of the secondary message may be less because of limited resources in the sending system.

10.17.2.9.4 For data sets with *Discrete* records, the format of each item is either ASCII (format 20) or binary (format 10). There is no requirement that all records be in the same format, but mixed record formats are not encouraged. Items with ASCII format should have only data characters. Characters which the sending system uses for control information (e.g., newline for a record terminator) should not appear. If an application finds it necessary to include these characters, then format 10, or Stream should be used.

10.17.2.10 *CLOSE Transaction* — This transaction terminates a data transfer and frees the *Handle* for future use. The primary message is sent by the receiving system and contains only the *Handle.* The sending system responds with a secondary message which has a return code.

10.17.2.10.1 The return code is one of the following:

OK.

ERROR: No open Data Set (i.e., incorrect *Handle*).

ERROR: Pending Transaction.

10.17.2.11 *Sending a Data Set* — Writing a data set is performed by requesting that the receiving system read it. The sending system initiates the SEND transaction to request that a data set be read. The receiving system is expected to perform the OPEN, READ, and CLOSE transactions to transfer the data set if it accepts the request. The time between the secondary message of the SEND and the primary message of the OPEN depends on the application.

10.17.2.12 *The SEND Transaction* — The primary message sent from the sending system to the receiving system contains the *Data Set Name.*

10.17.2.12.1 The secondary message contains the *Data Set Name* and a return code which is one of the following:

OK.

ERROR: Unknown Data Set Name.

ERROR: Try later (i.e., the system is busy).

10.17.2.13 *Error Recovery* — The receiving system may crash while a data set is open but no READ transaction is pending. The sending system will not be able to tell that this has happened because there is no timeout value defined between READ transactions. When the receiving system is restarted it may have forgotten which data sets were open. States in the two systems are now inconsistent.

10.17.2.14 *RESET Transaction* — The RESET transaction offers a way to resynchronize the two systems. When one system issues the primary message of the RESET transaction, it is informing the other system that any data sets which may have been open are to be closed. This applies to all data sets open between both systems. It is not necessary to issue CLOSE transactions for each individual data set because the RESET transaction is a global close.

10.17.2.15 Any equipment which uses Stream 13 must issue the RESET transaction as part of its initialization or bootstrap procedure. A host system must issue a RESET to equipment which uses Stream 13 during the initialization for that equipment.

10.17.2.16 *SECS-II Protocol Definition* — Figure 11 shows the state diagram for the sending system while a data set is being transferred. Each circle shows a possible state of the sending system. The names of these states are for reference only. They are not meant to suggest an implementation. The arrows show transitions due to SECS-II messages received or sent by the sending system.

10.17.2.16.1 In the initial state (Idle) handle X is not open. The states marked with an asterisk are those in which a transaction is outstanding. If the sending system receives any primary message from the receiving system with handle X during the time it is in these states, then the secondary message for that transaction will contain the "Pending Transaction" error code, but the original transaction for handle X will not be affected. Some states, especially the error states, may take zero time in some implementations. In these cases, the "Pending Transaction" error code would not be returned from those states.

10.17.3 *Formatted Data Sets* — Formatted data sets are data sets transferred in a standard format. Stream 13 provides a method for transferring data sets in a table format. A table has both attributes and content. The attributes of the table provide information about the data set as a whole, such as the date and time that it was last modified, its size, etc. The content of the table consists of column headers and rows. A row is an ordered list of table elements. A column refers to all table elements at a specific position within all rows, where each column is identified by a corresponding text string as a column header. The table elements in the 1st column position are used as an identifier for the row.

Stream,Function Name (Mnemonic)	*Direction*
S13,F0 Abort Transaction (S13F0)	S,H<->E
Description	
Same form as S1,F0.	
Structure	
Exception	

Stream,Function Name (Mnemonic)	*Direction*
S13,F1 Send Data Set Send (DSSS)	S,H<->E,reply
Description	
Sent by the sending system to request that the other system read a dataset.	
Structure	

`L,1`
`  1. <DSNAME>`	

Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S13,F2 Send Data Set Acknowledge (DSSA)	S,H<->E
Description	

Sent by the receiving system in response to Send Data Set Send.
 	
`1. <DSNAME>`
`2. <ACKC13>`	

Structure	
Exception	

The possible ACKC13 codes for this message are:
 	
`0 = O.K.`
`1 = ERROR:Try later.`
`2 = ERROR:Unknown Data Set Name.`	

Stream,Function Name (Mnemonic)	*Direction*
S13,F3 Open Data Set Request (DSOR)	S,H<->E,reply
Description	
Sent by the receiving system to open a data set for reading.	
Structure	

`L,3`
`  1. <HANDLE>`
`  2. <DSNAME>`
`  3. <CKPNT>`	

Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S13,F4 Open Data Set Data (DSOD)	S,H<->E

Description

Sent by the sending system in response to Open Data Set Request.

Structure

```
L,5
  1. <HANDLE>
  2. <DSNAME>
  3. <ACKC13>
  4. <RTYPE>
  5. <RECLEN>
```

Exception

The possible ACKC13 codes for this message are:

```
 0 = O.K.
 1 = ERROR:Try later.
 2 = ERROR:Unknown Data Set Name.
 3 = ERROR:Illegal Checkpoint value.
 4 = ERROR:Too many open Data Sets.
 5 = ERROR:Data set open too many times.
 9 = ERROR:Handle in Use.
10 = ERROR:Pending Transaction.
```

Stream,Function Name (Mnemonic)	*Direction*
S13,F5 Read Data Set Request (DSRR)	S,H<->E,reply

Description

Sent by the receiving system to read data from an open data set.

Structure

```
L,2
  1. <HANDLE>
  2. <READLN>
```

Exception

None

Stream,Function Name (Mnemonic)	*Direction*
S13,F6 Read Data Set Data (DSRD)	M,H<->E
Description	
Sent by the sending system in response to Read Data Set Request.	
Structure	

`L,4`
`  1. <HANDLE>`
`  2. <ACKC13>`
`  3. <CKPNT>`
`  4. L,n`
`       1. <FILDAT>`
`       .`
`       .`
`       n. <FILDAT>`	

Exception	
The possible item formats, number of items (n), and length of each FILDAT item (|th) are given by the following table. MaxItems is defined in Section 10.17.10.2.	

```
RTYPE              0 (Stream)          1 (Discrete)
Item Format        10 (binary)         10 (binary)
                                       or 20 (ASCII)
Maximum n          1                   MaxItems
Maximum n          1 (ACKC13 = 0)      1 (ACKC13 = 0)
                   0(any error)        0(any error)
Maximum |th        READLN              RECLEN
Maximum |th        0                   0

The possible ACKC13 codes for this message are:
 0 = O.K.
 6 = ERROR: No open Data Set
 7 = ERROR: Cannot Continue
 8 = ERROR: End of Data
10 = ERROR: Pending Transaction
```

Stream,Function Name (Mnemonic)	*Direction*
S13,F7 Close Data Set Send (DSCS)	S,H<->E,reply
Description	
Sent by the receiving system to close an open data set.	
Structure	

`L,1`
`  1. <HANDLE>`	

Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S13,F8 Close Data Set Acknowledge (DSCA)	S,H<->E
Description	
Sent by the sending system in response to Close Data Set Send (DSCS).	
Structure	

`L,2`
`  1. <HANDLE>`
`  2. <ACKC13>`
 	
`The possible ACKC13 codes for this message are:`
` 0 = O.K.`
` 6 = ERROR:No open Data Set.`
`10 = ERROR:Pending Transaction.`	

Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S13,F9 Reset Data Set Send (DSRS)	S,H<->E,reply
Description	
Sent by either system to close all open data sets.	
Structure	
`Header only`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S13,F10 Reset Data Set Acknowledge (DSRA)	S,H<->E
Description	
Sent in response to Reset Data Set Send.	
Structure	
`Header only`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S13,F11 Data Set Object Multi-Block Inquire (DSOMGI)	S,H<->E,reply
Description	
This message requests permission to send a multi-block data set. If the receiving system does not grant permission in the reply, the multi-block data set may not be sent. OBJSPEC is used to identify the data set object type and identifier and may include a destination. DATALENGTH represents the total message length, not the length of the data set.	
Structure	

`L,3`
`  1. <DATAID>`
`  2. <OBJSPEC>`
`  3. <DATALENGTH>`	

Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S13,F12 Data Set Object Multi-Block Grant (DSOMBG)	S,H<->E
Description	
This message grants or denies permission to send a multi-block data set.	
Structure	
`<GRANT>`	
Exception	
None	

Stream,Function Name (Mnemonic)	*Direction*
S13,F13 Table Data Send (TDS)	M,H<->E,reply

Description

This message allows the host and the equipment to exchange predefined datasets in a tabular format. The first element of every row is used to reference that row for all other elements. If S13,F13 is Multi-block, it must be preceded by the S13,F11/S13,F12 Inquire/Grant transaction.

Structure

```
L,8
  1. <DATAID>
  2. <OBJSPEC>
  3. <TBLTYP>
  4. <TBLID>
  5. <TBLCMD>
  6. L,n                        # of table attributes
       1. L,2
            1. <ATTRID$_1$>
            2. <ATTRDATA$_1$>
       .
       .
       n. L,2
            1. <ATTRID$_n$>
            2. <ATTRDATA$_n$>
  7. L,c                        # of column definitions
       1. <COLHDR$_1$>               1st column element description
       .
       .
       c. <COLHDR$_c$>               cth column element description
  8. L,r                        # of row definitions
       1. L,c$_1$                    # of entries per definition
            1. <TBLELT$_{11}$>         1st table element, 1st row
            .
            m. <TBLELT$_{1c1}$>        mth table element, 1st row
       .
       .
       r. L,c$_r$                    rth row definition
            1. <TBLELT$_{r1}$>         1st table element, rth row
           .
            m. <TBLELT$_{rcr}$>        mth table element, rth row
```

Exception

If OBJSPEC is a zero-length item, then the owner of the table is the receiver of the message. If r is zero, any existing table definition of the given type and id is to be deleted. Otherwise, c_1 may not be zero, and the value of c_1 shall be less than or equal to the value of c.

Stream,Function Name (Mnemonic)	*Direction*
S13,F14 Table Data Acknowledge (TDA)	S,H<->E

Description

This message is used to acknowledge the receipt of a table and to indicate any errors.

Structure

```
L,2
  1. <TBLACK>
  2. L,p
       1. L,2
            1. <ERRCODE$_1$>
            2. <ERRTEXT$_1$>
       .
       .
       p. L,2
            1. <ERRCODE$_p$>
            2. <ERRTEXT$_p$>
```

Exception

p = 0 if and only if TBLACK indicates no errors.

Stream,Function Name (Mnemonic)	*Direction*
S13,F15 Table Data Request (TDR)	M,H<->E,reply

Description

This message allows the host or the equipment to request part or all of a specific table. Either specific columns or specific rows may be requested, but not both at the same time. If S13,F15 is Multi-block, it must be preceded by the S13,F11/S13,F12 Inquire/Grant transaction.

Structure

```
L,7
  1. <DATAID>
  2. <OBJSPEC>
  3. <TBLTYP>
  4. <TBLID>
  5. <TBLCMD>
  6. L,p                       # of column definitions
       1. <COLHDR$_1$>              1st column element description
       .
       .
       p. <COLHDR$_p$>              pth column element description
  7. L,q
       1. <TBLELT$_1$>              1st row identifier
       .
       .
       q. <TBLELT$_q$>
```

Exception

If OBJSPEC is a zero-length item, then the owner of the table is the receiver of the message. Either p or q, or both, must be zero. If p = 0 and q = 0, all rows are requested; otherwise, only the specified columns, or the rows referenced by TBLELT, are requested.

Stream,Function Name (Mnemonic)	*Direction*
S13,F16 Table Data (TD)	M,H<->E

Description

This message is used to return data from the requested table.

Structure

```
L,6
  1. <TBLTYP>
  2. <TBLID>
  3. L,n                         # of table attributes
       1. L,2
            1. <ATTRID$_1$>
            2. <ATTRDATA$_1$>
       .
       .
       n. L,2
            1. <ATTRID$_n$>
            2. <ATTRDATA$_n$>
  4. L,c                         # of column definitions
       1. <COLHDR$_1$>              1st column element description
       .
       .
       c. <COLHDR$_c$>              cth column element description
  5. L,r                         # of row definitions
       1. L,c$_1$                   # of entries per definition
            1. <TBLELT$_{11}$>         1st table element, 1st row
            .
            .
            m. <TBLELT$_{1c1}$>        last table element, 1st row
       .
       .
       r. L,c$_r$                   rth row definition
            1. <TBLELT$_{r1}$>         1st table element, rth row
            .
            .
            m. <TBLELT$_{rcr}$>        mth table element, rth row
  6. L,2
       1. <TBLACK>
       2. L,p
            1. L,2
                 1. <ERRCODE$_{11}$>
                 2. <ERRTEXT$_{12}$>
            .
            p. L,2
                 1. <ERRCODE$_{p1}$>
                 2. <ERRTEXT$_{p2}$>
```

Exception

p = 0 if, and only if, TBLACK indicates no errors. The length c_{11} of a table row may not exceed the value of c.

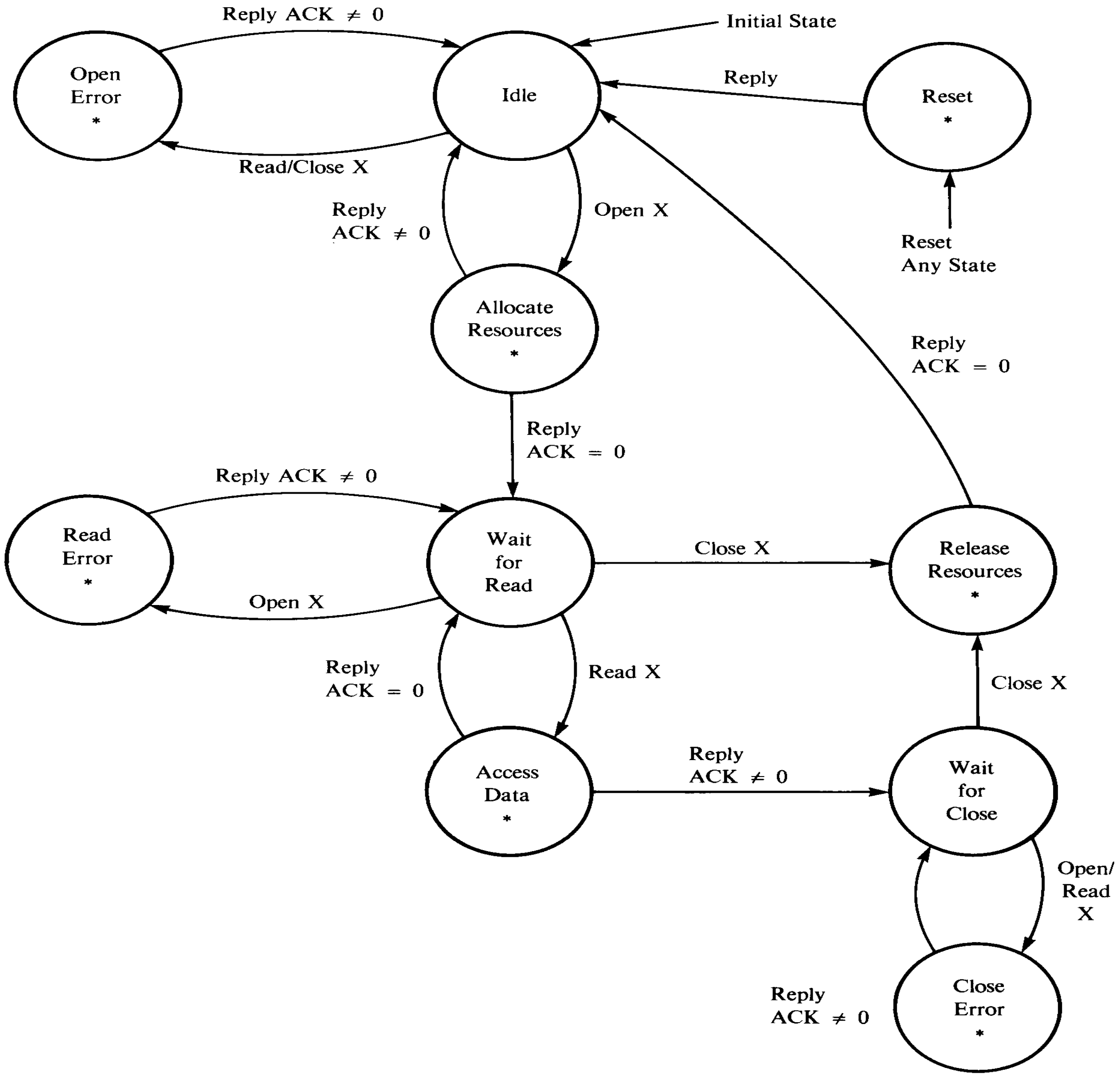

Figure 11
The Sending System's State Diagram During a Data Set Transfer
An Asterisk (*) marks the states where a primary message which uses handle X would result in a secondary message with the error code for "Pending Transaction."

10.18 *Stream 14 Object Services* — The functions in this stream are used for generic functions concerning objects, including obtaining information about objects and setting values for an object.

Stream,Function Name (Mnemonic)	*Direction*
S14,F0 Abort Transaction (S14F0)	S,H<->E
Description	
Same form as S1,F0.	
Structure	
Exception	

Stream,Function Name (Mnemonic)	*Direction*
S14,F1 GetAttr Request (GAR)	S,H<->E,reply

Description

This message is used to request a set of specified attributes for one or more objects. It consists of an “object specifier” for the owner of the target objects (the objects of interest), the target object type, a list of identifiers of the target objects, a filter (a list of qualifying relationships) that limits the target objects of interest to those that meet all of the qualifications in and the specific attributes whose values are requested.

The object specifier provides a specification of the owner of the target object(s). It contains a sequence of hierarchical object relationships. Each element of the object specifier identifies a specific object instance that is the superior of the following object instance in the sequence. The last object instance in the sequence is in a hierarchical relationship to the target objects. The target object type designates the type of the target object, and the list of object identifiers indicates the specific instance of that type that are of interest. The target type may be omitted only if object identifiers are unique across all object types and the list of identifiers is not empty.

The object filter is an optional list of qualifications, each of which provides a condition to be applied to the object instances of interest. Each qualification objects of interest are those that meet all of the specified qualifications.

The attribute relationship quantifier is a logical binary relationship ATTRRELN_i that the specified qualifying value ATTRDATA_i has to the corresponding attribute of each instance of the desired object type(s). The objects that are to be qualified with this filter have an attribute value V_i such that the statement “ATTRDATA_i ATTRRELN_i V_i” is TRUE. If ATTRRELN_i is omitted, the relationship of equality is intended.

For ASCII attribute values ATTRDATA_i, the characters for question mark “?” and asterisk “*” are used as “wild characters” to provide filtering for certain object types. The character “?” may be used in any attribute or key attribute value with an ASCII format to represent “any single character” and may be repeated. The asterisk character “*” may be similarly used to represent a variable-length string, including a null string. The string “*x” represents a string of any length that ends in “x”, the string “x*” represents any string that begins with “x”, and the string “*” represents any string of any non-zero length. The comparison for text characters is case insensitive.

Equipment is not required to support wild characters in particular, or attribute filters in general.

Structure

```
L,5
  1. <OBJSPEC>
  2. <OBJTYPE>
  3. L,i                         i = identifiers of the object instances requested
       1. <OBJID1>
       .
       .
       i. <OBJIDi>
  4. L,q                         q = # object qualifiers to match
       1. L,3
            1. <ATTRID1>
            2. <ATTRDATA1>
            3. <ATTRRELN1>
       .
       .
       q. L,3
            1. <ATTRIDq>
            2. <ATTRDATAq>
            3. <ATTRRELNq>
  5. L,a                         a = # attributes requested
       1. <ATTRID1>
       .
       .
       a. <ATTRIDa>
```

Exception

If OBJSPEC is a zero-length item, no object specifier is provided. If i = 0,only the filter is to be applied. If q = 0, no filter is specified. If both i and q = 0, information for all instances of the objects are requested. If a = 0, all attributes are requested.

Stream,Function Name (Mnemonic)	*Direction*
S14,F2 GetAttr Data (GAD)	M,H<->E
Description	
This message is used to transfer the set of requested attributes for the specified object(s). The order of attributes is retained from the primary message.	
Structure	

```
L,2
  1. L,n                         n = number of objects
       1. L,2
            1. <OBJID1>
            2. L,a               a = number of attributes
                 1. L,2
                      1. <ATTRID1>
                      2. <ATTRDATA1>
                 .
                 .
                 a. L,2
                      1. <ATTRIDa>
                      2. <ATTRDATAa>
       .
       .
       n. L,2
            1. <OBJIDn>
            2. L,b               b = number of attributes
                 1. L,2
                      1. <ATTRID1>
                      2. <ATTRDATA1>
                 .
                 .
                 b. L,2
                      1. <ATTRIDb>
                      2. <ATTRDATAb>
  2. L,2
       1. <OBJACK>
       2. L,p                    p = number of errors reported
            1. L,2
                 1. <ERRCODE1>
                 2. <ERRTEXT1>
            .
            .
            p. L,2
                 1. <ERRCODEp>
                 2. <ERRTEXTp>
```

Exception
If OBJSPEC is a zero-length item, no object specifier is provided. If n = 0, no objects matched the specified filter. If p = 0, no errors were detected.

Stream,Function Name (Mnemonic)	*Direction*
S14,F3 SetAttr Request (SAR)	S,H<->E, reply

Description

This message is used to request that a given set of attributes be assigned specified values for all objects of the specified type and exactly matching the specified attribute requirements. Certain attributes may not be changed through the interface. For a description of filters, see S14,F1.

Structure

```
L,4
  1. <OBJSPEC>
  2. <OBJTYPE>
  3. L,i                         i = number of object instances requested
       1. <OBJID1>
       .
       .
       i. <OBJIDi>
  4. L,n                         n = # attribute settings
       1. L,2
            1. <ATTRID1>
            2. <ATTRDATA1>
       .
       .
       n. L,2
            1. <ATTRIDn>
            2. <ATTRDATAn>
```

Exception

If OBJSPEC is a zero-length item, no object specifier is provided.

Stream,Function Name (Mnemonic)	*Direction*
S14,F4 SetAttr Data (SAD)	M,H<->E
Description	
This message is used to acknowledge that the attributes for the specified objects have been set as requested or to indicate an error for each attribute value that was not set as requested. The order of attributes is retained from the primary message.	
Structure	

```
L,2
  1. L,i                         i = number of objects requested
       1. L,2
            1. <OBJID_1>
            2. L,n               n = number of attributes set.
                 1. L,2
                      1. <ATTRID_1>
                      2. <ATTRDATA_1>
                 .
                 .
                 n. L,2
                      1. <ATTRID_n>
                      2. <ATTRDATA_n>
       .
       .
       i. L,2
            1. <OBJID_i>
            2. L,n
                 1. L,2
                      1. <ATTRID_1>
                      2. <ATTRDATA_1>
                 .
                 .
                 n. L,2
                      1. <ATTRID_n>
                      2. <ATTRDATA_n>
  2. L,2
       1. <OBJACK>
       2. L,p                    p = number of errors reported
            1. L,2
                 1. <ERRCODE_1>
                 2. <ERRTEXT_1>
            .
            .
            p. L,2
                 1. <ERRCODE_p>
                 2. <ERRTEXT_p>
```

Exception
If n = 0 for any object, the object was not found. If p = 0, no errors were detected.

Stream,Function Name (Mnemonic)	*Direction*
S14,F5 GetType Request (GTR)	S,H<->E,reply
Description	
This message is used to request the types of objects owned by an object. This is an operation performed on an object type rather than on object instances. Wild characters “?” and “*” may be used as a filter for object types. Equipment is not required to support wild characters.	
Structure	
`<OBJSPEC>`	
Exception	
If OBJSPEC is a zero-length item, no object specifier is provided.	

Stream,Function Name (Mnemonic)	*Direction*
S14,F6 GetType Data (GTD)	
Description	
Structure	

```
L,2
  1. L,n                           n = number of object types
       1. <OBJTYP1>
       .
       .
       n. <OBJTYPn>
  2. L,2
       1. <OBJACK>
       2. L,p                      p = number of errors reported
            1. L,2
                 1. <ERRCODE1>
                 2. <ERRTEXT1>
            .
            .
            p. L,2
                 1. <ERRCODEp>
                 2. <ERRTEXTp>
```

Exception
If n = 0, there are no owned object types. If p = 0, no errors were detected.

Stream,Function Name (Mnemonic)	*Direction*
S14,F7 GetAttrName Request (GANR)	S,H<->E,reply
Description	
This message is used to request the names of the attributes of specified types of owned objects. This is an operation performed on an object type rather than on object instances. Wild characters “?” and “*” may be used as a filter for object types. Equipment is not required to support wild characters.	
Structure	

```
L,2
  1. <OBJSPEC>
  2. L,n                           n = # of object types
       1. <OBJTYP1>
       .
       .
       n. <OBJTYPn>
```

Exception
If OBJSPEC is a zero-length item, no object specifier is provided.

Stream,Function Name (Mnemonic)	*Direction*
S14,F8 GetAttrName Data (GAND)	M,H<->E
Description	
This message contains the names of the attributes of the requested objects.	
Structure	

```
L,2
  1. L,n                         n = number of object types
       1. L,2
            1. <OBJTYP_1>
            2. L,a               a = number of attributes
                 1. <ATTRID_1>
                 .
                 .
                 a. <ATTRID_a>
       .
       .
       n. L,2
            1. <OBJTYP_n>
            2. L,b               b = number of attributes
                 1. <ATTRID_1>
                 .
                 .
                 b. <ATTRID_b>
  2. L,2
       1. <OBJACK>
       2. L,p                    p = number of errors reported
            1. L,2
                 1. <ERRCODE_1>
                 2. <ERRTEXT_1>
            .
            .
            p. L,2
                 1. <ERRCODE_p>
                 2. <ERRTEXT_p>
```

Exception
If OBJSPEC is a zero-length item, no objects matched the specified filter. If p = 0, no errors were detected.

Stream,Function Name (Mnemonic)	*Direction*
S14,F9 Create Object Request (COR)	M,H<->E,reply

Description

This message is used to request an object owner to create an object instance. OBJSPEC specifies the object owner.

Structure

```
L,3
  1. <OBJSPEC>
  2. <OBJTYPE>
  3. L,a                         a = # attributes requested
       1. L,2
            1. <ATTRID$_1$>
            2. <ATTRDATA$_1$>
       .
       .
       a. L,2
            1. <ATTRID$_a$>
            2. <ATTRDATA$_a$>
```

Exception

If OBJSPEC is a null-length item, no object specifier is provided. If a = 0, no specific attribute settings are requested for the new object.

Stream,Function Name (Mnemonic)	*Direction*
S14,F10 Create Object Acknowledge (CAO)	M,H<->E

Description

This message is used to acknowledge the success or failure of creating the new object specified. If successful, OBJSPEC is the object specifier of the new object. The list of attributes returned is dependent upon the type of object specified.

Structure

```
L,3
  1. <OBJSPEC>
  2. L,b                         b = number of attributes returned
       1. L,2
            1. <ATTRID$_1$>
            2. <ATTRDATA$_1$>
       .
       .
       b. L,2
            1. <ATTRID$_b$>
            2. <ATTRDATA$_b$>
  3. L,2
       1. <OBJACK>
       2. L,p                    p = number of errors reported
            1. L,2
                 1. <ERRCODE$_1$>
                 2. <ERRTEXT$_1$>
            .
            .
            p. L,2
                 1. <ERRCODE$_p$>
                 2. <ERRTEXT$_p$>
```

Exception

If OBJSPEC is a null-length item, no object was created. If b = 0, no attributes of the new object are returned. If p = 0, no errors were detected.

Stream,Function Name (Mnemonic)	*Direction*
S14,F11 Delete Object Request	S,H<->E,reply

Description

This message is used to request that the object specified in OBJSPEC be deleted. The list of attribute settings depends upon the type of object to be deleted.

Structure

```
L,2
  1. <OBJSPEC>
  2. L,a                         n = # attribute settings
       1. L,2
            1. <ATTRID_1>
            2. <ATTRDATA_1>
       .
       .
       a. L,2
            1. <ATTRID_a>
            2. <ATTRDATA_a>
```

Exception

If n = 0, no attribute settings are provided.

Stream,Function Name (Mnemonic)	*Direction*
S14,F12 Delete Object Acknowledge (DOA)	M,H<->E

Description

This message is used to acknowledge the success or failure of deleting the object specified. The list of attributes returned is dependent upon the type of object to be deleted.

Structure

```
L,2
  1. L,b                         n = number of attributes returned
       1. L,2
            1. <ATTRID_1>
            2. <ATTRDATA_1>
       .
       .
       b. L,2
            1. <ATTRID_b>
            2. <ATTRDATA_b>
  2. L,2
       1. <OBJACK>
       2. L,p                    p = number of errors reported
            1. L,2
                 1. <ERRCODE_1>
                 2. <ERRTEXT_1>
            .
            .
            p. L,2
                 1. <ERRCODE_p>
                 2. <ERRTEXT_p>
```

Exception

If n = 0, no attribute values are returned. If p = 0, no errors were detected.

Stream,Function Name (Mnemonic)	*Direction*
S14,F13 Object Attach Request (OAR)	M,H<->E,reply
Description	
This message is sent by a supervisor to request the object specified in OBJSPEC to attach or reattach itself to the requestor.	
Structure	

```
L,2
  1. <OBJSPEC>
  2. L,a                         a = # attribute settings
       1. L,2
            1. <ATTRID_1>
            2. <ATTRDATA_1>
       .
       .
       a. L,2
            1. <ATTRID_a>
            2. <ATTRDATA_a>
```

Exception	
If a = 0, no attribute settings are provided.	

Stream,Function Name (Mnemonic)	*Direction*
S14,F14 Object Attach Acknowledge (OAA)	M,H<->E
Description	
This message is used to acknowledge the success or failure of the requested attachment. If successful, a non-zero token shall be returned for the supervisor's use in subsequent communications with the attached object.	
Structure	

```
L,3
  1. <OBJTOKEN>
  2. L,b                         b = number of attributes
       1. L,2
            1. <ATTRID_1>
            2. <ATTRDATA_1>
       .
       .
       b. L,2
            1. <ATTRID_b>
            2. <ATTRDATA_b>
  3. L,2
       1. <OBJACK>
       2. L,p                    p = number of errors reported
            1. L,2
                 1. <ERRCODE_1>
                 2. <ERRTEXT_1>
            .
            .
            p. L,2
                 1. <ERRCODE_p>
                 2. <ERRTEXT_p>
```

Exception	
OBJTOKEN is zero if and only if p is non-zero. If b = 0, no attribute values are returned. If p = 0, no errors were detected.	

Stream,Function Name (Mnemonic)	*Direction*
S14,F15 Attached Object Action Request (AOAR)	M,H<->E,reply
Description	
This message is used by a supervisor (only) to request an attached object to perform an action.	
Structure	

```
L,4
  1. <OBJSPEC>
  2. <OBJCMD>
  3. <OBJTOKEN>
  4. L,a                         a = # attribute settings
       1. L,2
            1. <ATTRID$_1$>
            2. <ATTRDATA$_1$>
       .
       .
       a. L,2
            1. <ATTRID$_a$>
            2. <ATTRDATA$_a$>
```

Exception
If a = 0, no attribute settings are provided.

Stream,Function Name (Mnemonic)	*Direction*
S14,F16 Attached Object Action Acknowledge (AOAA)	M,H<->E
Description	
This message is used to acknowledge the success or failure of an action requested by a supervisor.	
Structure	

```
L,2
  1. L,b                         b = number of attributes
       1. L,2
            1. <ATTRID$_1$>
            2. <ATTRDATA$_1$>
       .
       .
       b. L,2
            1. <ATTRID$_b$>
            2. <ATTRDATA$_b$>
  2. L,2
       1. <OBJACK>
       2. L,p                    p = number of errors reported
            1. L,2
                 1. <ERRCODE$_1$>
                 2. <ERRTEXT$_1$>
            .
            .
            p. L,2
                 1. <ERRCODE$_p$>
                 2. <ERRTEXT$_p$>
```

Exception
If p = 0, no errors were detected.

Stream,Function Name (Mnemonic)	*Direction*
S14,F17 Supervised Object Action Request (SOAR)	S,H<->E,reply

Description

This message is used to request a supervisor to have a supervised attached object perform an action. OBJSPEC specifies the supervisor, and TARGETSPEC specifies the attached object.

Structure

```
L,4
  1. <OBJSPEC>
  2. <OBJCMD>
  3. <TARGETSPEC>
  4. L,a                         a = number of attribute settings
       1. L,2
            1. <ATTRID$_1$>
            2. <ATTRDATA$_1$>
       .
       .
       a. L,2
            1. <ATTRID$_a$>
            2. <ATTRDATA$_a$>
```

Exception

If a = 0, no attribute settings are provided.

Stream,Function Name (Mnemonic)	*Direction*
S14,F18 Supervised Object Action Acknowledge (SOAA)	M,H<->E

Description

This message is used to acknowledge the success or failure of an action requested of a supervisor.

Structure

```
L,2
  1. L,b                         b = number of attributes
       1. L,2
            1. <ATTRID$_1$>
            2. <ATTRDATA$_1$>
       .
       .
       b. L,2
            1. <ATTRID$_b$>
            2. <ATTRDATA$_b$>
  2. L,2
       1. <OBJACK>
       2. L,p                    p = number of errors reported
            1. L,2
                 1. <ERRCODE$_1$>
                 2. <ERRTEXT$_1$>
            .
            .
            p. L,2
                 1. <ERRCODE$_p$>
                 2. <ERRTEXT$_p$>
```

Exception

If b = 0, no attributes are returned. If p = 0, no errors were detected.

Stream,Function Name (Mnemonic)	*Direction*
S14,F19 Generic Service Request (GSR)	M,H->E,reply

Description

The host requests an object to perform the specified service with its associated parameters. If multi-block, it shall be preceded by the S14F23/F24 Multi-Block Inquire/Grant transaction. DATAID is given uniquely to each message. OPID is uniquely specified to identify delayed completion information for time consuming service. OPID could be zero if and only if the service cannot take a long time.

Structure

```
L,5
  1. <DATAID>
  2. <OPID>
  3. <OBJSPEC>
  4. <SVCNAME>
  5. L,m                         # of parameter groups
       1. L,2
            1. <SPNAME1>          service parameter 1 name
            2. <SPVAL1>           service parameter 1 value
       2. L,2
            1. <SPNAME2>          service parameter 2 name
            2. <SPVAL2>           service parameter 2 value
       .
       .
       m. L,2
            1. <SPNAMEm>          service parameter m name
            2. <SPVALm>           service parameter m value

If a specific value of SPNAME is defined to have a SPVAL defined as a LIST, it shall
always be a LIST.  If the SPVAL that is associated to that specific value of SPNAME is
defined to be anything other than LIST, it will result in a format error.
```

Exception

A zero length list, m = 0, indicates that no parameter groups are sent with the service request. OBJSPEC can be a null length item if no object provide the services is defined in the standards which are referred to and it is assumed that “Equipment” is delegated and handled as if it is an object.

Notes:

1. If some service parameters are attributes of the specified object, service parameter name-value pair, that is SPNAME and SPVAL, is actually attribute id-data pair, that is ATTRID and ATTRDATA. An example of parameter part in the message format could be interpreted as below.

```
L,m
  1. L,2
       1. <SPNAME1>
       2. <SPVAL1>
  2. L,2
  .
  .
  k. L,2
       1. <ATTRIDk>
       2. <ATTRDATAk>
  .
  .
  m. …
```

2. If SPVAL is a LIST, the items that make up that list shall take on one of the following forms: (1) a list of items with an identical format, (2) a LIST of SPNAME, SPVAL pairs, as illustrated below. When SPVAL is actually ATTRDATA, even if it is a LIST, it or its parts are not required to expand into lower level items if their names have not been formally named in the corresponding SEMI standard.

```
A)  L,2
      1. <SPNAMEa>
      2. L,m
```

```
           1. <SPVAL_a1>
           2. <SPVAL_a2>
           .
           .
           m. <SPVAL_am>
B)  L,2
      1. <SPNAME_b>
      2. L, n
            1. L,2
                 1. <SPNAME_bl>
                 2. <SPVAL_bl>
            .
            .
            n. L,2
                 1. <SPNANE_bn>
                 2. <SPVAL_bn>
```