SCB Architecture - Move Mode

Insert the hardtab labeled "Move Mode" here
and discard this page.

Move Mode

Figures

Chapter 1. Overview

The Move mode supports the delivery of control information between cooperating entities using a variable-length control element. This control element can contain requests, replies, errors, or event notifications for an entity in another adapter or system unit, or for entities in the same adapter or system unit.

At the processing level, operation of the Move mode is similar to that of the Locate mode (client entities build requests, requests are delivered to server entities, server entities build replies, and replies are delivered to client entities).

However, in the Move mode, a client entity can be in an adapter as well as in a system unit, which means an entity acting as a client in an adapter can send requests to a server entity in another adapter or to the system unit. This capability provides a peer-to-peer relationship between entities independent of their role (client or server) or their physical location (adapter or system unit). A server can be in the system unit, which means that the system unit can receive requests from an adapter.

The Move mode defines a method of delivery that permits a varying number of control elements to be moved from an entity in one unit to a correspondent entity in another unit through a pair of delivery pipes. An entity uses one pipe for sending control elements and the other pipe for receiving control elements. Each pipe can contain requests, replies, errors or event notifications for a specific entity in another unit, or for different entities in the same unit.

The Move mode defines the pipe-related protocols and the services that support the control element delivery.

Figure 1-1 on page 1-3 shows how various levels of delivery-support relate to each other and to the various control areas in I/O space and shared memory. This figure also shows that there is send-interface logic and receive-interface logic between the delivery support and the client and server entities. These interfaces provide the support necessary to adapt the entity-interface requirements to those of delivery support, taking into account the specific local operating

environment. There is, however, no unit-to-unit protocol associated with the send-interface logic and receive-interface logic.

The architecture definition of the control areas, control elements, services, and protocols used in Move mode are presented in the remainder of this manual.

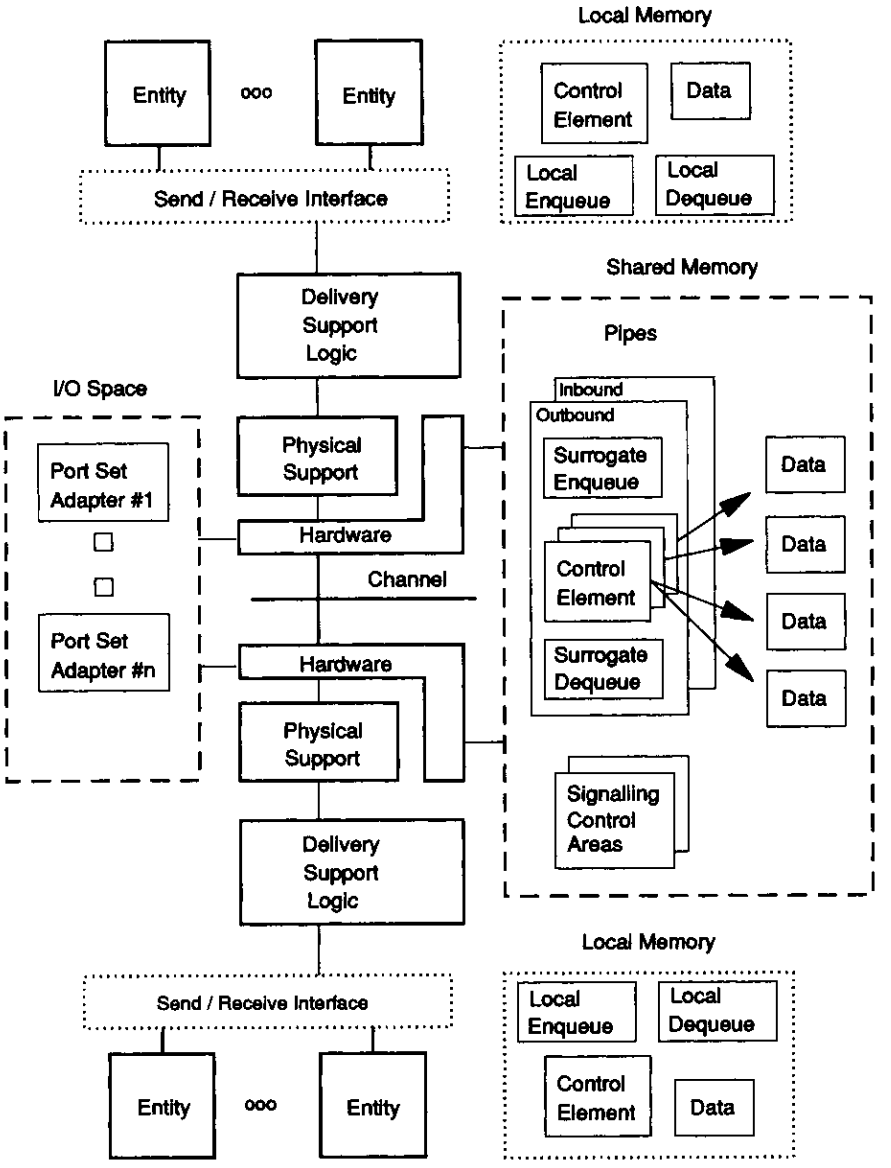

Figure 1-1. Control Element Delivery Structure

Operational Characteristics

The following are some of the key operational characteristics of the Move mode.

- Request and Reply Support

  A request and reply protocol is used to support the delivery of control elements. An adapter can send requests to the system unit and receive replies from the system unit because an adapter can contain a client entity or a server entity. This support includes a means to:

  - Send one or more requests and receive one or more replies in response.
  - Correlate a request with one or more replies.
  - Identify the source and destination of both requests and replies.
  - Send and receive unsolicited information.

- Bidirectional Delivery

  Requests and replies can flow in both directions.

- Peer-to-Peer Delivery Support

  An entity in a system unit or adapter can send requests directly to, and receive replies directly from the entities in a system unit or another adapter attached to the channel. This means an entity in an adapter can send requests and receive replies directly from an entity in another adapter without having to go through a system unit.

- Variable Length Requests and Replies

  Requests and replies are exchanged using variable-length control elements. The control elements allow a variable number of parameters of any size to be passed between client and server entities. This provides a simple way of tailoring control elements to the task to be performed rather than using fixed-length control elements.

- Delivery Pipes

  Request and reply control elements are moved between entities in a source unit and entities in a destination unit using a delivery pipe. A delivery pipe is similar to a first-in, first-out queue, which makes the control elements appear to be contiguous. This allows for the chaining of multiple control elements into a single request.

- Asynchronous Operation

  The use of a delivery pipe allows multiple control elements to be delivered and processed asynchronously by the entities within each unit (system unit or adapter).

- Full Duplex

  A pipe for each direction of delivery between each unit (system unit or adapter) allows control elements to be delivered in one direction independent of the flow of control elements in the other direction.

- Multiplexed

  Control elements for multiple entities in the same destination unit can be found within the same delivery pipe.

  The delivery pipe supports the intermixing of requests and replies as well as other types of control elements (errors and events) within the same pipe for different entities.

- Continuous Running

  The structure of the delivery pipe allows control element delivery to appear as a continuous stream of control elements flowing between client and server entities in the source and destination units. There are elements of the architecture that assist in the management of this continuous stream of control elements:

  - Suspending the delivery of control elements at the entity-to-entity level.
  - Notifying the destination entity that a control element is available.
  - Synchronizing the exchange of control elements between source and destination entities.

- Shared Memory

  Shared memory allows control structures that convey control elements, as well as the control structures supporting the delivery pipes, to be placed in the memory of either an adapter or system unit. This greatly enhances the ability to make cost and performance trade-offs when designing advanced function adapters.

- Reduction of Interrupts

  It is possible to reduce or eliminate the use of physical interrupts between units (system unit or adapter). Each delivery pipe can specify whether interrupts are used and what conditions require the use of interrupts.

Peer to Peer Relationships

In a peer-to-peer relationship, entities performing the roles of clients and servers are not restricted to being in an adapter or a system unit, but can be located in either place. Therefore, control elements can be delivered directly between a system unit and an adapter, or between any two adapters on the channel.

To operate in a peer-to-peer relationship, the delivery support must allow requests and replies to flow in either direction and allow control elements for entities in the same destination unit to be intermixed on the same delivery pipe. In control-element delivery, this is supported by having independent delivery of control elements in either direction, and by allowing clients to be located in system units or adapters, and servers to be located in system units or adapters. A peer-to-peer relationship also requires support in both system units and adapters to resolve contention when two or more system units or adapters attempt to deliver control elements to the same destination at the same time.

The term *peer-to-peer* must be qualified by the level of support being discussed and not used as a unit-to-unit term without qualification.

The peer-to-peer delivery support has a pair of delivery pipes for each unit-to-unit pair with entities that communicate with each other. An example of this is shown in the following figure. The labels R1, R2, and so on, on the client and server entities indicate the entity pairings. The dotted lines indicate the delivery support portion in each unit.

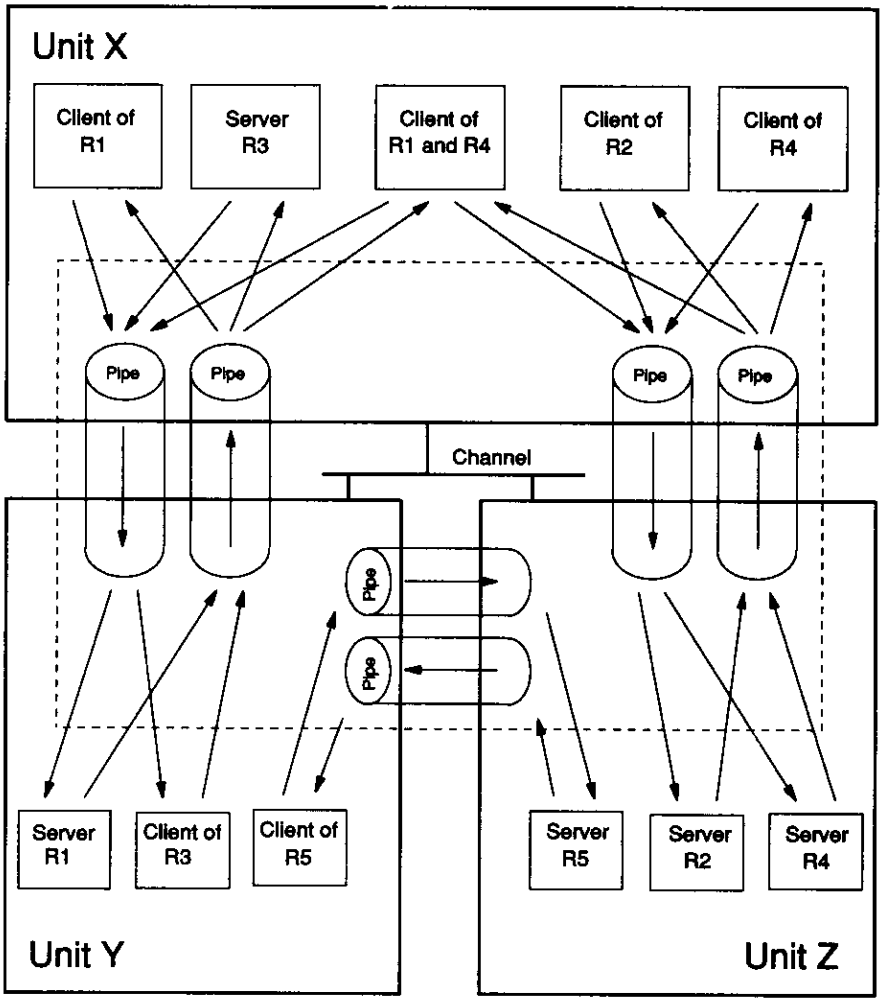

Figure 1-2. Peer to Peer Delivery Model

Queueing Control Elements

Figure 1-3 on page 1-10 shows an example of the flow of a request through the delivery pipe to the specific server queues. Each entity pair is responsible for ensuring that a pending-receive indication is sent so that one entity cannot block others from using the pipe. If the pending-receive indication is not sent, the delivery-level support can discard the control element and notify the entity that sent the control element. Queueing by the server is an optional function, which is performed within the processing level.

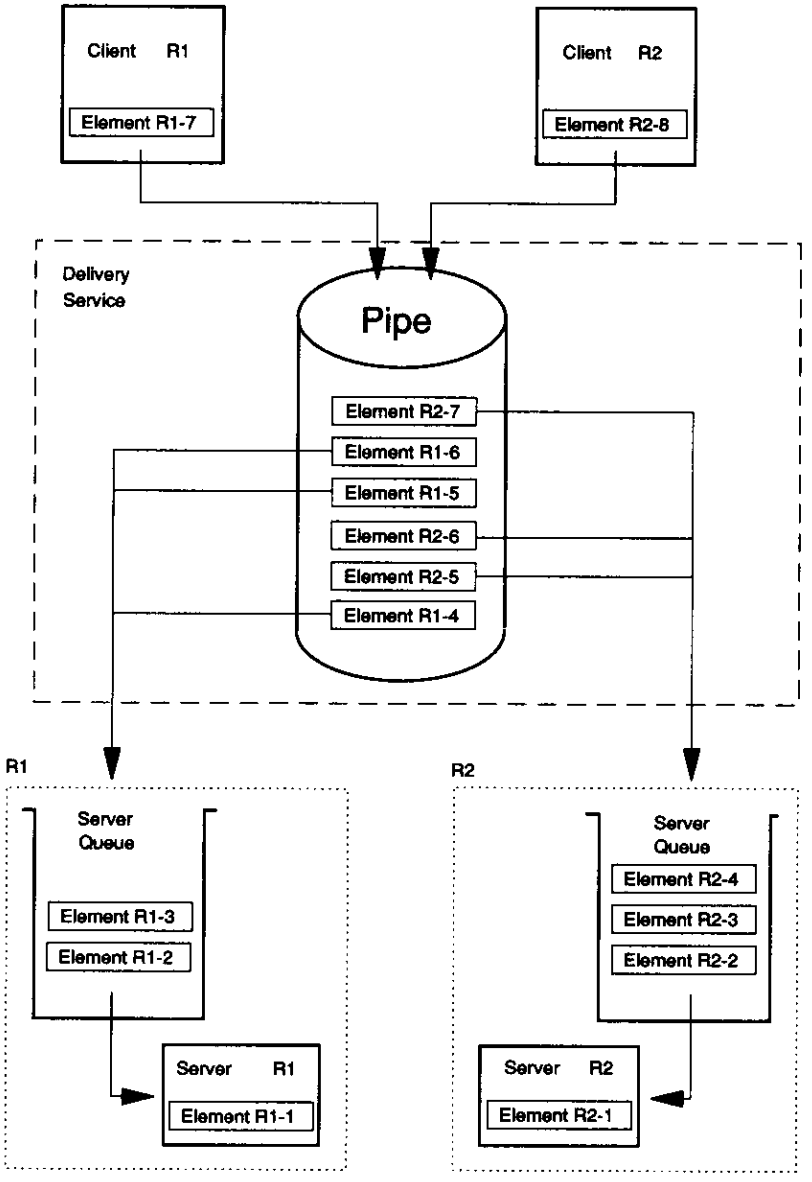

Figure 1-3. Handling Control Elements

Control Element Delivery

The following figure shows the overall flow of a request or a reply between the server entity (device attachment or processing software) in an adapter and the client entity in another adapter or system unit.

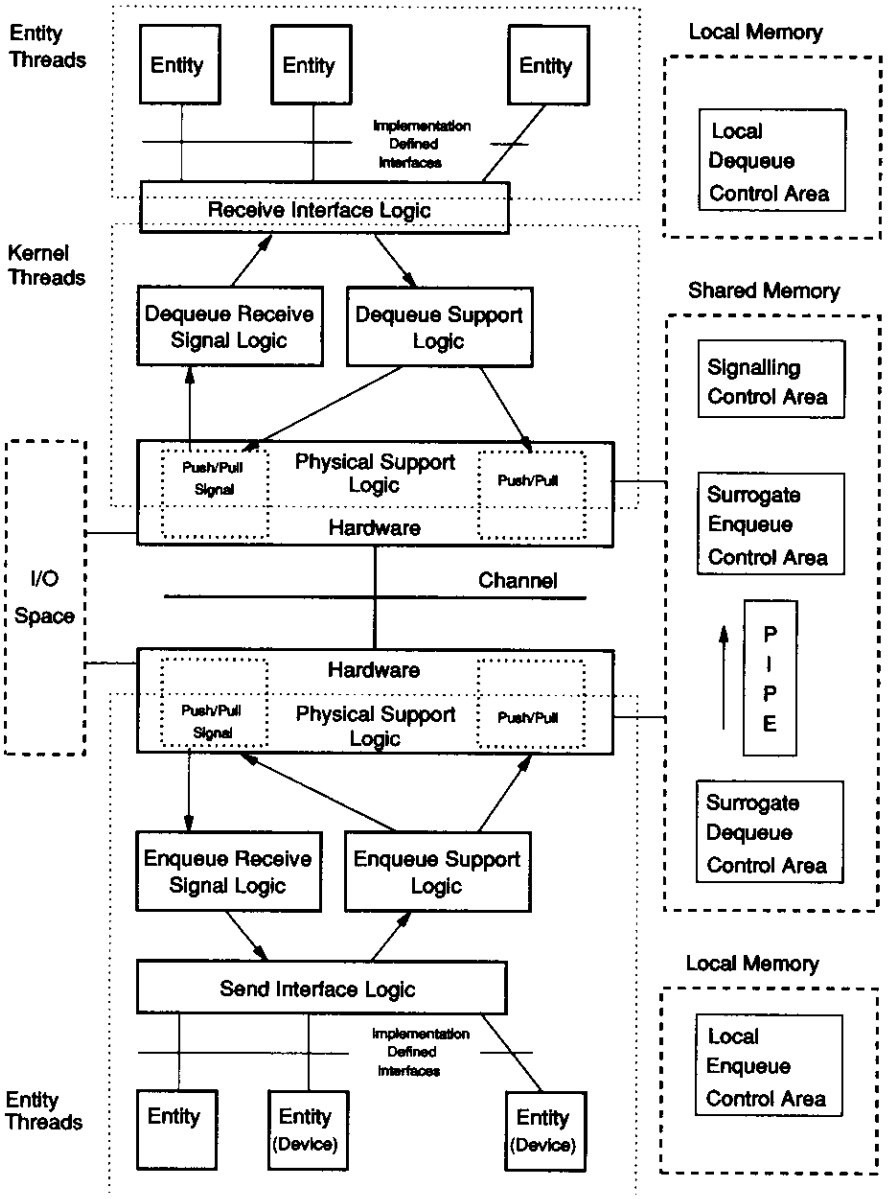

Figure 1-4. Control Element Delivery Flow

The following describes the flow and logic for delivering a request to the server. The flow and logic for delivering a reply to a client is identical but inverted.

The send-interface logic is responsible for sending control elements from an entity in one unit to a correspondent entity in the same or different unit. This interface enforces the policies of the send operation that are consistent with a particular implementation and operating environment. For example, it defines the form that the call takes, and how waits and sharing of the delivery pipe are handled. The send-interface logic also handles the changing of addresses from the local address space to the address space in shared memory (this logic is determined by, and specific to, the implementation).

The send-interface logic calls the enqueue logic with the address and count of the number of control elements to be sent. Using the information in the Destination and Length fields of each control element, the enqueue logic places the control elements one at a time into the appropriate delivery pipe and updates the associated pointers. This continues until all the control elements are placed into the delivery pipe or the pipe becomes full.

If the pipe becomes full, the enqueue logic stops enqueueing control elements, updates the Status fields in the Local and Surrogate Enqueue Control areas, sends a signal to the dequeue logic at the other end of the delivery pipe, and returns a pipe-full condition to the send-interface logic

If the delivery pipe is empty when a control element is enqueued, the enqueue logic updates the Status fields in the Local and Surrogate Enqueue Control areas, and sends a signal to the dequeue logic at the other end of the delivery pipe.

The enqueue logic sends a signal to the dequeue logic by first setting the dequeue state-change indicator in the Signalling Control area assigned to the unit and then invoking the signalling services of the physical-support logic.

If the system unit is being signalled, then the physical-support logic activates the appropriate interrupt-request line. If the adapter is being signalled, then the physical-support logic writes the appropriate attention code into the Attention port of the adapter.

Note: Each state-change indicator (enqueue, dequeue, and management) is assigned to a separate byte in the Signalling Control area. This allows each state-change indicator to be written separately and requires only the posting of the appropriate delivery-support or management-support logic when a state-change indicator is set.

On the receive side, the receive-interface logic is responsible for providing the interface between the dequeue logic and the entities that receive control elements from correspondent entities. This interface enforces the policies of the receive operation that are consistent with a particular implementation and operating environment. For example, it defines:

- What form the call takes
- How waits are handled
- Whether a read pending is indicated to remove control elements or a queue of the removed control elements is maintained.

The receive-interface logic also handles the transformation of addresses and the movement of control elements from the shared memory address space to the local address space of the receiving entity or queue (these are determined by, and specific to, the implementation).

The receive-interface logic initiates the removal of control elements from a delivery pipe by calling the dequeue logic. If there is a control element in the delivery pipe, the dequeue logic calls the receive-interface logic with the address of the control element to be dequeued. The receive-interface logic, using information in the control element (Destination and Length fields), determines where the control element should be sent. Depending on the policy of the receive-interface logic, it can choose to:

- Route it directly to the destination entity
- Place it on a queue maintained by the receive-interface logic for the destination entity
- Return it to the originating entity
- Discard it.

When the receive-interface logic returns to the dequeue logic, the information in the Local and Surrogate Control areas associated with the delivery pipe is updated and the process is repeated until all control elements in the delivery pipe are removed. At this time, the

dequeue logic returns to the receive-interface logic at the point of the original invocation.

Receiving a signal sent by the enqueue logic at the other end of the delivery pipe can also initiate removing control elements from a delivery pipe. In this case, the interrupt logic associated with the physical-support logic examines all Signalling Control areas and, based on the state-change indicators, posts the appropriate dequeue receive-signal logic.

The posting, in effect, allows the receive-interface logic and dequeue logic to run under the kernel thread. It also allows the logic to synchronize with the entity thread portion of the receive-interface logic for the entity indicated by the Destination field of each control element removed from the delivery pipe.

I/O Address Space

Multiple ports are defined in the architecture for the physical interface. In this context, a *Port* describes a byte or set of contiguous bytes located in I/O address space.

Since several adapters can be present on the channel, the I/O addresses are shown as offsets from the base I/O address of the adapter. The base I/O address used by each adapter is determined during system configuration.

Physical Interface Support

The physical interface supports the delivery of control elements between a system unit and an adapter as well as between adapters. It uses the following subset of the Locate mode ports:

- Attention port
- Subsystem Control port
- Command Busy/Status port.

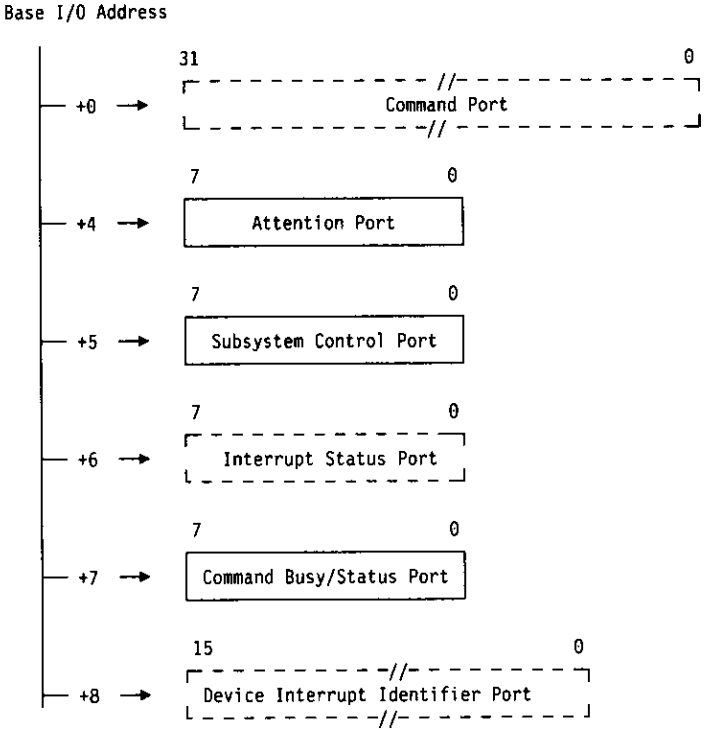

Figure 1-5. I/O Address Space Map - Physical Interface

The Command, Interrupt Status, and Device Interrupt Identifier ports are present, but are not used by Move mode after the physical adapter has reached operational state.

Attention Port

The Attention port is an 8-bit port used by the physical-support logic in a system unit or adapter to signal, or request the attention of, another adapter. Writing to the Attention port causes the adapter to be interrupted and indicates that signalling information has been placed into the Signalling Control area assigned to the system unit or adapter writing to the Attention port. (See "Signal" on page 2-5 for additional information on the Signalling operation and "Signalling Control Area" on page 1-29 for a description of the Signalling Control areas.)

The format and content of the Attention port are shown in the following figure.

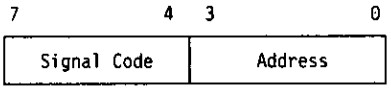

Figure 1-6. Attention Port

The Signal Code field (bits 7-4) must be set to hex D, and the Address field (bits 3-0) must be set to 0.

Subsystem Control Port

The Subsystem Control port is an 8-bit port that is used by the system unit to provide direct hardware control of several adapter functions. This port contains control bits that are used to:

- Enable interrupts from the adapter
- Enable the bus-master operations by the adapter
- Reset the adapter.

The following figure shows the format and content of the Subsystem Control port and is followed by a description of the individual bits.

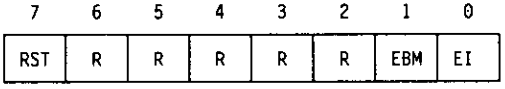

Figure 1-7. Subsystem Control Port

Bit 7 The reset bit (RST) is used to provide a hardware-controlled reset of the adapter and its associated resources and attached devices. When set to 1, all subsystem and device activities on the adapter are halted and the subsystem is placed in the reset state.

Bits 6 – 2 These bits are reserved.

Bit 1 The enable-bus-master bit (EBM) is used to control bus-master operations from the adapter. Setting this bit to 1 enables the adapter to initiate bus-master operations. Setting this bit to 0 disables the initiation of bus-master operations by the adapter. The enable-bus-master bit is initialized to 0 when the adapter is powered-on or reset.

Note: The enable-bus-master bit is primarily used to debug hardware. However, it can also be used to

force the adapter to halt adapter-initiated channel activity.

Bit 0 The enable-interrupt bit (EI) is used to control the interrupts generated by requests from the adapter. Setting this bit to 1 allows the adapter to send interrupt requests to the system unit. Setting the bit to 0 prevents the adapter from sending physical interrupts. The enable-interrupt bit is set to 0 when the adapter is powered-on or reset.

Command Busy/Status Port

The Command Busy/Status port is an 8-bit port that has three functions:

- It indicates that an interrupt to the system unit is pending.
- It indicates that a signal to the adapter is pending when used in conjunction with the Attention port.
- It indicates that a reset request is in progress in the adapter when used in conjunction with the Subsystem Control port.

The following figure shows the format and content of the Command Busy/Status port and is followed by a description of the individual bits.

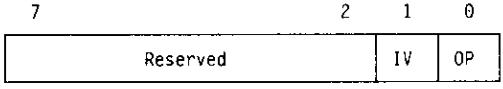

Figure 1-8. Command Busy/Status Port

Bits 7 – 2 These bits are reserved.

Bit 1 The interrupt-valid bit (IV) indicates that there is a valid interrupt request pending from this adapter to the system unit. It is set as part of the signalling logic and is reset when the physical-support logic in the system unit reads the Command Busy/Status port.

Bit 0 The operation-pending bit (OP) indicates the current state of a signalling operation or the progress of a hardware reset of the adapter (for example, the toggling of the reset (RST) bit in the Subsystem Control port).

Memory Address Space

Shared memory is used to hold control elements (request, reply, error, and event) within the in-bound and out-bound pipes. It also contains:

- Surrogate Control areas
- Signalling Control areas
- Data areas.

The format, content, and use of control elements are described in "Control Elements" on page 1-32. This section describes the format, content, and use of the following control areas:

- Delivery pipe areas
- Enqueue Control areas
- Dequeue Control areas
- Signalling Control areas.

Delivery Pipe

A delivery pipe is a control area maintained in shared memory. It is used to support the exchange of control elements between cooperating entities in different units (system units or adapters). The pipe space is managed as a first-in first-out (FIFO) stream in which entities in one unit insert control elements at one end and entities in another unit take the control elements out at the other end.

A system unit or adapter has a pair of pipes for each adapter or system unit it exchanges control elements with. One pipe handles the flow of control elements in one direction; the other pipe handles the flow of control elements in the opposite direction.

Each pipe represents a circular queue and contains an area to hold the actual control elements. The structure of this area is shown in the following figure.

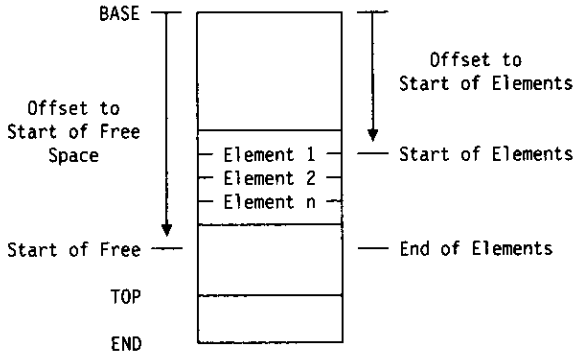

Figure 1-9. Pipe Area

Additional control areas are used to manage the operation of the pipe. For each pipe there is a Local Enqueue Control area, Surrogate Enqueue Control area, Local Dequeue Control area, and Surrogate Dequeue Control area. The relationship of these control areas to the pipe and to each other is shown in Figure 1-10 on page 1-21.

Note: The Surrogate Enqueue Control area is write only to the enqueue logic and read only to the dequeue logic at the other end of the pipe. The Surrogate Dequeue Control area is read only to the enqueue logic and write only to the dequeue logic at the other end of the pipe.

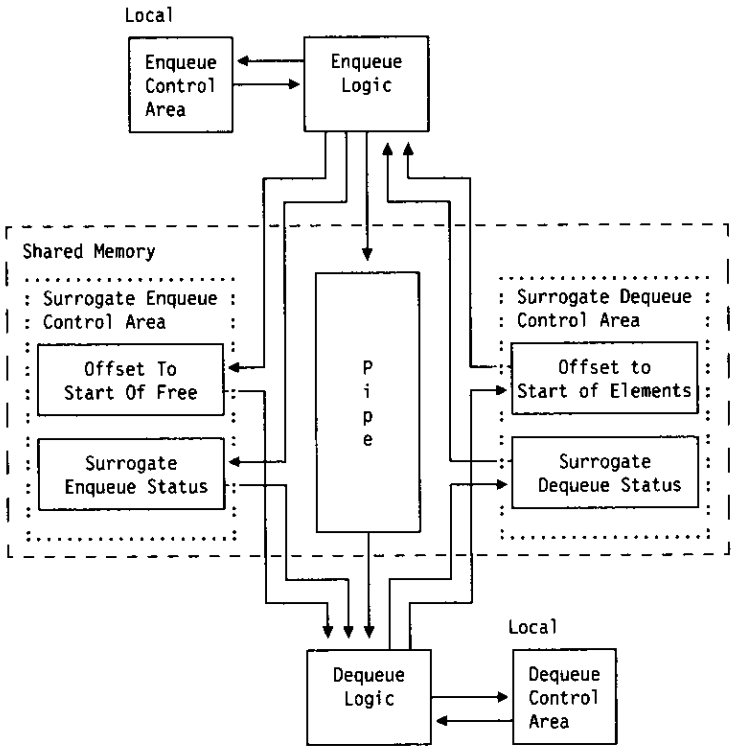

Figure 1-10. Control Areas Associated with a Single Pipe

Local Enqueue Control Area

Within a given unit, there is a Local Enqueue Control area for each pipe. Each control area is identical in structure.

The Local Enqueue Control area is constructed and maintained in either local or shared memory from information provided in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Fields within each Local Enqueue Control area identify the location of the pipe, indicate its current status (full, empty, or wrap), and provide state information identifying the starting and ending offsets of control elements in the pipe. The following shows the structure and format of the Local Enqueue Control area and describes each of the fields.

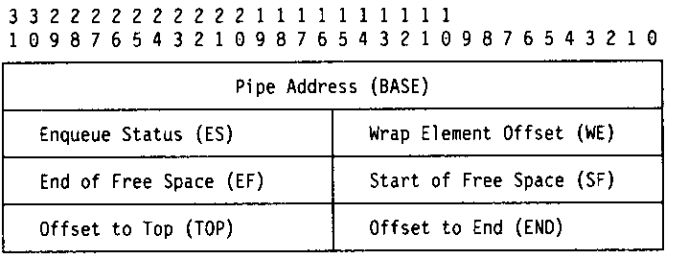

Figure 1-11. Local Enqueue Control Area Structure

The Pipe Address field is a 32-bit, doubleword-aligned field containing the 32-bit physical address of the area in shared memory where the circular queue of control elements is maintained.

The Enqueue Status field is a 16-bit, word-aligned field containing bits used to maintain current status information of the pipe as perceived by the local enqueue logic. The format and content of this field are shown below and are followed by a description of each of the bits.

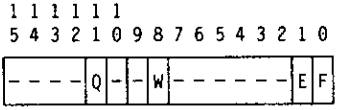

Figure 1-12. Enqueue Status Field

Bits 15–12 These bits are reserved.

Bit 11 The queued bit (Q) indicates that a control element has been successfully placed into the pipe.

Bits 10–9 These bits are reserved.

Bit 8 The wrap bit (W) indicates that a control element would not fit into the space remaining between the start-of-free space (SF) and the top-of-pipe space (TOP). The enqueue logic toggles the current setting of this bit. The bit is initially set to 0 during initialization or after management services re-establish the synchronization of both ends of a delivery pipe.

Bits 7–2 These bits are reserved.

Bit 1 The empty bit (E) is set to reflect the setting of the empty bit in the Surrogate Dequeue Status field.

Bit 0 The full bit (F) indicates whether the pipe is full. The enqueue logic sets this bit to 1 (full) when it tries to place a control element into the pipe and there is insufficient space. The enqueue logic resets this bit each time it successfully places a control element into the pipe. This bit is also reset to 0 (not full) during initialization or after re-establishing the synchronization of the pipe.

Note: The queued and empty bits are implementation specific and are shown here to help in understanding the enqueue algorithm that is presented later.

The Wrap Element Offset field is a 16-bit, word-aligned field containing the offset, in bytes, from the pipe address (BASE) to the location of the wrap control element in the pipe. This field is updated each time a wrap control element is placed into the pipe.

The End of Free Space field (EF) is a 16-bit, word-aligned field containing the offset, in bytes, to the end-of-free space (TAIL) in the pipe. This field represents the end of the available space for placing control elements into the pipe. This field is updated each time the enqueue logic is called. It is checked against the Start of Elements field in the Surrogate Dequeue Control area to insure control elements that have not been dequeued are not overwritten.

The Start of Free Space field (SF) is a 16-bit, word-aligned field containing the offset, in bytes, to the start-of-free space (HEAD) in the pipe. It is the location for the next control element that is placed into the pipe. This field is updated each time a control element is placed into the pipe.

The Offset to Top field (TOP) is a 16-bit, word-aligned field containing the offset, in bytes, to the end of the pipe that is used for holding control elements. This field is set at initialization time and is a copy of information contained in the configuration record.

The Offset to End field (END) is a 16-bit, word-aligned field containing the offset, in bytes, to the end of the pipe. This field represents the

physical size of the pipe when allocating space in shared memory for the pipe and is based on product requirements.

Note: Offset to Top and Offset to End fields ensure that sufficient space is reserved at the end of the pipe to hold a wrap control element.

Surrogate Enqueue Control Area

There are two fields in the Surrogate Enqueue Control area associated with the enqueue operations of a pipe: Surrogate Start of Free (SSF) and Surrogate Enqueue Status (SES). In this context, *Surrogate* means that the information in these two fields is a copy of the information found in the Local Enqueue Control area. This surrogate information is used by the dequeue logic, at the other end of the pipe, in conjunction with the information from the Local Enqueue Control area to manage the distributed aspects of the pipe.

Surrogate Enqueue Control areas are constructed and maintained in shared memory from information provided in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Surrogate Start of Free: The Surrogate Start of Free field is a 16-bit, word-aligned field containing the offset in bytes from the pipe address (BASE) to the location in the pipe where the enqueue logic places the next control element. The format of the field is shown in the following figure.

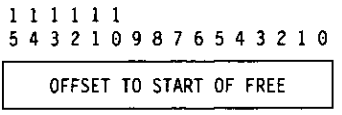

Figure 1-13. Surrogate Start of Free Field

The Surrogate Start of Free field is updated by the enqueue logic after each control element is placed into the pipe. The enqueue logic writes to the field and the dequeue logic reads from the field.

Surrogate Enqueue Status: The Surrogate Enqueue Status field is a 16-bit, word-aligned field that contains bits used to inform the dequeue logic of the current state of the pipe, as perceived by the enqueue logic. The enqueue logic writes to this field, and the dequeue logic reads from this field.

Note: The enqueue logic updates all of the bits in the Surrogate Enqueue Status field each time a control element is place into the pipe.

The following shows the format and content of the field and describes each of the bits.

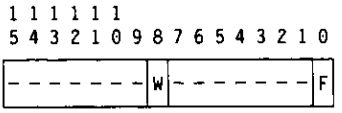

Figure 1-14. Surrogate Enqueue Status Field

Bits 15 – 9 These bits are reserved.

Bits 8, 0 These bits are copies of the bits in the Enqueue Status field. (See the description of the Enqueue Status field on page 1-22.)

Bits 7 – 1 These bits are reserved.

Local Dequeue Control Area

Like the Local Enqueue Control area described previously, there is one Local Dequeue Control area in a given unit for each incoming pipe. They are all identical in structure.

The Local Dequeue Control area is constructed and maintained in either local or shared memory from information provided in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Fields within the Local Dequeue Control area identify the physical location of the pipe, indicate the current status of the pipe (full, empty, or wrap), and provide state information identifying the starting and ending offsets of control elements in the pipe. The following

figure shows the structure of the Local Dequeue Control area. It is followed by a description of the content and use of each of its fields.

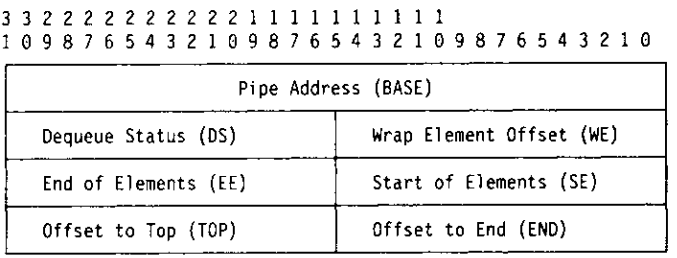

Figure 1-15. Local Dequeue Control Area Structure

The Pipe Address field (BASE) is a 32-bit, doubleword-aligned field, containing the 32-bit physical address of the area in shared memory where the circular queue of control elements is maintained.

The Dequeue Status field (DS) is a 16-bit field containing bits used to maintain status information about the current state of the pipe, as perceived by the local dequeue logic. The following figure shows the format and meaning of the bits and is followed by a description of each bit.

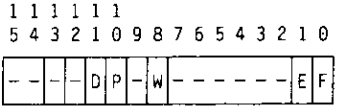

Figure 1-16. Dequeue Status Field

Bits 15—12 These bits are reserved.

Bit 11 The dequeued bit (D) is an internal bit that indicates a control element has been removed from the pipe. It is reset to 0 prior to exiting the dequeue logic.

Bit 10 The pre-empt bit (P) is an internal bit that indicates no more control elements will be removed from the pipe, even if the pipe is not empty. This bit is set and reset by the dequeue logic. It is set depending upon the value returned from the receive-interface logic.

Bit 9 This bit is reserved.

Bit 8 The wrap bit (W) indicates that a wrap control element has been placed in the pipe because there was insufficient space between the top of the pipe and the start of free space for the control element. The dequeue logic toggles the current setting of this bit after removing a wrap control element. The bit is initially set to 0 during initialization or after re-establishing the synchronization of the pipe.

Bits 7 – 2 These bits are reserved.

Bit 1 The empty bit (E) indicates whether the pipe is empty. The dequeue logic sets this bit to 1 when it determines that the pipe is empty. The dequeue logic resets this bit to 0 (not empty) when it determines that the empty bit in the Surrogate Enqueue Status area is set to 0. The bit is also set to 1 (empty) during initialization or after re-establishing the synchronization of the pipe.

Bit 0 The full bit (F) is set to reflect the setting of the full bit in the Surrogate Enqueue Status field.

Note: The dequeued, pre-empt, and full bits are implementation specific and are shown here to help in understanding the dequeue algorithm that is presented later.

The Wrap Element Offset field is a 16-bit, word-aligned field containing the offset, in bytes, from the pipe address to the location of the wrap control element in the pipe.

The End of Elements field is a 16-bit, word-aligned field containing the offset, in bytes, to the end of the control elements in the pipe. This points to the byte immediately following the last control element in the pipe. This field is updated each time the dequeue logic is called; it is a copy of information maintained as an offset in the Start of Free field in the Surrogate Enqueue Control area.

The Start of Elements field is a 16-bit, word-aligned field containing the offset, in bytes, to the start of the next control element in the pipe. This field is updated each time a control element is removed from the pipe.

The Offset To Top field is a 16-bit, word-aligned field containing the offset, in bytes, to the end of the usable area in the pipe for holding control elements. This field is set at initialization time using

information maintained in the configuration record. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

The Offset To End field is a 16-bit, word-aligned field containing the offset, in bytes, to the physical end of the pipe. This field represents the physical size of the pipe when allocating space in shared memory for the pipe and is based on product requirements.

Note: Offset to Top and Offset to End fields ensure that sufficient space is reserved at the end of the pipe to hold a wrap control element.

Surrogate Dequeue Control Area

There are two fields in the Surrogate Dequeue Control area associated with the dequeue operations of a pipe: Surrogate Start of Elements (SSE) and Surrogate Dequeue Status (SDS). In this context, *Surrogate* means that the information in these two fields is a copy of information found in the Local Dequeue Control area. This surrogate information is used by the enqueue logic, at the other end of the pipe, in conjunction with information from the Local Enqueue Control area to manage the distributed aspects of the pipe.

Surrogate Start of Elements: The Surrogate Start of Elements field (SSE) is a single 16-bit, word-aligned field containing the offset, in bytes, from the pipe address to the location in the pipe where the dequeue logic finds the next control element to be removed.

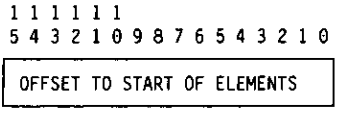

Figure 1-17. Surrogate Start of Elements Field

The Surrogate Start of Elements field is updated by the dequeue logic each time a control element is removed from the pipe. The Surrogate Start of Elements field is read, but never written to by the enqueue logic.

Surrogate Dequeue Status: The Surrogate Dequeue Status field is a single 16-bit, word-aligned field containing bits used to inform the enqueue logic of the current state of the pipe, as perceived by the dequeue logic. The format and content of the field is shown in the following figure. It is followed by a description of the meaning and use of each of its bits.

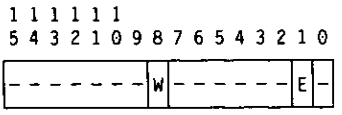

Figure 1-18. Surrogate Dequeue Status Field

Bits 15 – 9 These bits are reserved.

Bits 8, 1 These bits are copies of the bits in the Local Dequeue Status field. (See the description of the Local Dequeue Status field on page 1-26.)

Bits 7 – 2 These bits are reserved.

Bit 0 This bit is reserved.

The Surrogate Dequeue Control areas are constructed and maintained in shared memory from information provided in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Signalling Control Area

A system unit or adapter that receives signals from other units maintains a Signalling Control area in shared memory. It assigns a specific area within this Signalling Control area to each unit that will be signalling it (see Figure 1-19 on page 1-30).

These Signalling Control areas serve two purposes: they provide the means for identifying the source of the signal or interrupt and the means for indicating the reason for the signal or interrupt.

The location of these Signalling Control areas is provided in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details.)

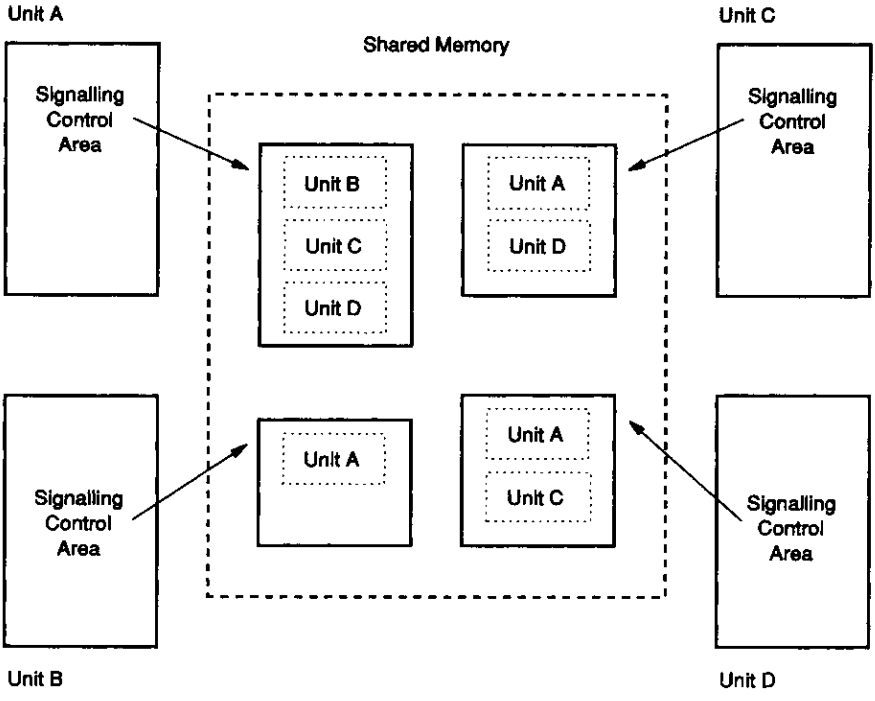

Figure 1-19. Signalling Control Areas

The following figure shows the structure of the Signalling Control area. It is followed by a description of the content and use of each of its fields.

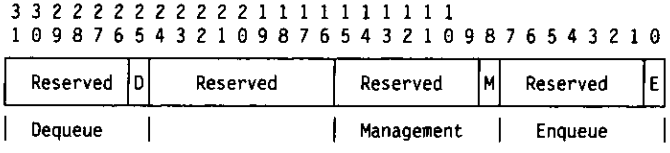

Figure 1-20. Signalling Control Area Structure

The Enqueue field is an 8-bit field; bit 0 indicates that a change in the state of the pipe is being signalled to the enqueue logic by the delivery-support logic at the other end of the pipe. The reason for the state change can be determined by examining the contents of the Surrogate Dequeue Status field associated with the pipe. Some state changes that can cause setting bit 0 of this field are:

- A control element has been placed into a previously empty delivery pipe.
- A control element could not be placed into a delivery pipe due to a full condition.
- A control element has been placed into a delivery pipe.
- A request to re-establish synchronization for the control structures at the other end of the delivery pipe.

Except for the last item, all of the above are optional and are specified in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Bits 7 – 1 of the Enqueue field are reserved.

The Management field is an 8-bit field; bit 0 is used to indicate a change in the state of the delivery logic at one end of the delivery pipe which requires the attention of the management logic at the other end of the delivery pipe. An example of such a state change is the timer expiration.

Bits 7 – 1 of the Management field are reserved.

The Dequeue field is an 8-bit field; bit 0 indicates that a change in the state of the pipe is being signalled to the dequeue logic by the delivery-support logic at the other end of the pipe. The reason for the state change can be determined by examining the contents of the Surrogate Enqueue Status field associated with the pipe. Some state changes that can cause setting bit 0 of this field are:

- A control element has been removed from a previously full delivery pipe.
- The last control element has been removed from a delivery pipe resulting in an empty condition.
- A control element was removed from a delivery pipe.

- A request to re-establish synchronization for the control structures to the other end of the delivery pipe has been requested.

Except for the last item, all of the above are optional and are specified in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Bits 7 – 1 of the Dequeue field are reserved.

In addition to identifying the reason for a signal, the Signalling Control area is also used to identify the source of the signal. The system unit and adapters can organize the Signalling Control area as a contiguous table. Each entry in the table would correspond to a Signalling Control area. This allows the location of each Signalling Control area to be assigned sequentially at initialization time, when configuration records are exchanged and updated (see "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Control Elements

In the Move mode, control elements are used to exchange control information and data between client and server. Some fields of a control element are used by the delivery service, as well as the client and server, while other fields contain information meaningful only to the client and server.

Note: Control elements are not used in Locate mode. In the Move mode, control elements must be aligned in doubleword boundaries.

Basic Structure

Each control element contains three components: type, length, and value. The type component is used to identify the format of the control element. The length component specifies the length of the control element. The value component contains information used by both the delivery service and the cooperating peer entities to

coordinate the exchange and processing of command and control information.

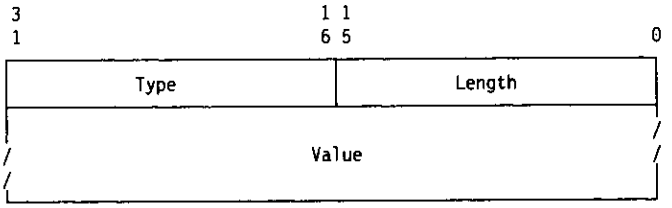

Figure 1-21. Basic Control Element Structure

Type Field

The type component of the control element contains a single field, the Format Identifier field (FID). This field is a 16-bit, word-aligned field used to specify the format and content of the architected portion of the control element. Control elements with a format-identifier value of 0 identify control elements defined for the Move mode.

Length Field

The length component of the control element contains a single field, the Length field. This field is a 16-bit, word-aligned field used to specify the total length of the control element. The length is specified in bytes and includes two bytes for the Length field, and a variable number of bytes for the Value field.

Value Field

The value component of the control element is variable in length and contains information whose structure and content is determined by the value of the Format Identifier field.

General Structure

Move mode control elements are identified by a format identifier of 0. The value component of these control elements consists of a fixed-length header and a variable-length body. The header part is common to all the Move mode control elements and contains the following fields:

- 16-bit Common Indicators field

- 16-bit Reserved field
- 16-bit Source field
- 16-bit Destination field
- 32-bit Correlation field.

The body consists of 0 or more variable-length Entity-to-Entity fields. The presence, format, content, and use of these fields are determined by the content of the Common Indicators field.

The Format Identifier and Length fields along with the content of the Value field are used by the delivery service, as well as the client and server. The remaining field, the Entity-to-Entity field, contains information meaningful only to the client and server.

The following describes, the format and content of each of these fields as it pertains to the delivery service. It also describes a set of common control elements that can be used to set up and manage the exchange of data between client and server.

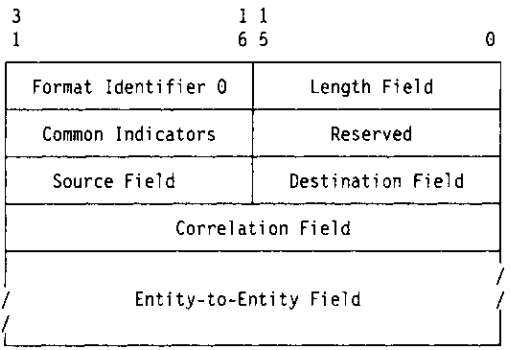

Figure 1-22. Control Element Structure - FID 0

Common Indicators Field

The Common Indicators field is a 16-bit, word-aligned field used to identify a control element and indicate how to interpret the remaining fields within the basic control element. The format and content of the Control Indicators field is illustrated in the following figure.

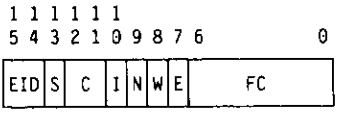

Figure 1-23. Control Element Common Indicators Field

The Common Indicators field consists of the following bits and subfields:

Bits 15, 14 The Element Identifier (EID) is a 2-bit subfield used to identify the control element type. The value in this subfield identifies one of four possible control element types:

ID	Description
00	The request is a control element sent by a client to a server asking it to perform a unit of work on its behalf. It allows the transfer of data and control information between client and server.
01	The reply is a control element sent by a server in response to certain request control elements. Not all request control elements are answered by replies, only those that request information.
10	The event is a control element that contains information about the progress of a request, the state of a previous request, or the status of a client or server. Events are unsolicited and can be sent by either a server or a client.
11	The error is a control element that provides error information about a previous request, or about the status of a server or delivery agent.

Bit 13 The suppress-reply bit (S) indicates whether the client wants the server to return a reply control element when the processing of this request control element has been completed successfully. When set to 1, the reply is suppressed and no reply control element is returned. When set to 0, the entity-to-entity protocol determines whether a reply control element is returned. The

suppress-reply bit is only valid in a request control element.

Bits 12, 11 The chaining bits (C) indicate a group of two or more control elements which represent a single unit of work to be processed in the order in which they appear. The following indicates how the chaining bits are interpreted. The chaining bits are only valid in a request control element.

ID	Description
00	No chaining (each control element is a separate unit of work)
01	First control element in a chain
11	Intermediate control element in a chain
10	Last control element in a chain.

Bit 10 The indirect bit (I) indicates whether the Entity-to-Entity field of a control element contains the parameters for the specified function, or just the length of and a pointer to the area in shared memory where the actual parameters are stored. When set to 1, the Entity-to-Entity field contains the length of and a pointer to the area containing the parameters. The indirect bit is only valid in a request control element.

Bit 9 The notification bit (N) indicates whether a client has requested notification by the server when the processing of this control element begins. When set to 1, a notification is returned to the client in an event control element when processing begins. The notification bit is only valid in a request control element (see "Notification and Wait" on page 4-11 for additional information).

Bit 8 The wait bit (W) indicates whether a client has requested the server to wait before processing this request control element, or any other request control elements from this client. The server suspends processing of the control element from that client until that client instructs the server to resume processing. When set to 1, all request control element processing is suspended, pending the receipt of a resume event control element. The wait bit is

only valid in a request control element (see "Notification and Wait" on page 4-11 for additional information).

Bit 7 The expedite bit (E) identifies the control element that the client wants the server to process as soon as possible (before any other control elements that might be waiting to be processed). When this bit is set to 1, the control element is expedited. The expedite bit is only valid in a request control element.

Bits 6 – 0 The Function Code subfield (FC) is a 7-bit subfield used to identify the function to be performed and indicate how to interpret the contents of the Entity-to-Entity field. Each of the Function Codes are described in detail in "Function Codes" on page 1-39.

Source Field

The Source field is a 16-bit, word-aligned field used to identify the originator or source of a control element. It consists of an 8-bit unit identifier and an 8-bit entity identifier (see Figure 1-24 on page 1-38).

The unit identifier in the Source field contains an identifier used to indicate which physical unit (system unit or adapter) the entity sending the control element is located. The value assigned to the unit identifier is determined by the delivery service during initialization. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for additional information.)

The entity identifier in the Source field contains an identifier used to indicate which of the possible 256 entities in the source unit is the originator of the control element. The value assigned to the entity identifier is determined by unit-level management and is implementation dependent.

Destination Field

The Destination field is a 16-bit, word-aligned field used to identify the target or destination of a control element. Like the Source field, it is a structured field containing an 8-bit unit identifier and an 8-bit entity identifier (see Figure 1-24 on page 1-38).

The unit identifier part of the Destination field contains an identifier used to indicate the location of the destination entity (system unit or

adapter). The value assigned to the unit identifier is determined by the delivery service during initialization. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for additional information.)

The entity identifier part of the Source field contains an identifier used to indicate which of the possible 256 entities in the destination unit is the recipient of the control element. The value assigned to the entity identifier is determined by unit-level management and is implementation dependent.

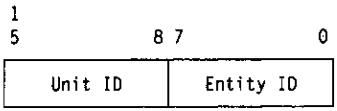

Figure 1-24. Control Element Source and Destination Fields

Correlation Field

The Correlation field is a 32-bit, word-aligned field used to provide an identifier for associating or correlating a reply control element, error control element, or an event control element with a previous request control element. The format, content, and use of the Correlation field are determined by the entity that originates the request control element. The field can contain a sequence number, the address of a control block, or the address of a data buffer. The content of the Correlation field is returned to the originator of a request in a reply, error, or event control element.

Entity to Entity Field

The Entity-to-Entity field of a control element is a variable-length field used to hold information such as status, parameters, or data required by the clients and servers to perform the operation identified by the function code in the Type field. The length, format, and type of information the field contains varies depending on the function specified. The information in the Entity-to-Entity field is meaningful only to the clients and servers and is defined as part of the entity-to-entity protocol.

Function Codes

Within the Common Indicators field of a Move mode control element, a function code is used to identify not only the operation to be performed but also to indicate how the contents of the Entity-to-Entity field should be interpreted. The high-order bit of the function code is used to distinguish between commonly-used function codes and implementation-defined function codes. When the high-order bit (6) is set to 1, it indicates that the remaining bits (5 – 0) contain an implementation-defined function code. When set to 0, bit 6 indicates that the remaining bits (0 – 5) contain one of the following commonly-used function codes.

Function Code	Name
1	Reserved
2	Initialize
3	Reserved
4	Read
5	Read List
6	Read Immediate
7	Write
8	Write List
9	Write Immediate
10	Execute List
11	Mark
12	Cancel
13	Reset
14	Read Configuration
15	Diagnose
16	Reserved
17	Resume
18	Notify
19	Inform
20-31	Reserved
32-39	Reserved
40	Reserved
41-62	Reserved
63	Wrap

Figure 1-25. Architected Function Codes in the Common Indicators Field

The resume, notify, inform, and wrap function codes are used exclusively with the event control element. "Event Control Elements" on page 1-62 provides a detailed description of these function codes. All other function codes are used with request, reply, and error control elements and are described in the next section.

Request, Reply, and Error Control Elements

The architecture defines a number of control elements for sending and receiving requests and replies. The following section describes these control elements. Each control element provides a set of primitives that can be used to create the specific protocol required for the task at hand. The rules determining protocol requirements such as the number, type, order of exchange, and required responses are determined by the two cooperating entities. This becomes the user-defined processing-level protocol.

Initialize

The Initialize function is used in request, reply, and error control elements. In a request control element, it invokes the Initialize function of the server. The Source, Destination, and Correlation Identifier fields, and Entity-to-Entity field, containing any additional parameters required by the server Initialize function, are passed to the server in the request control element.

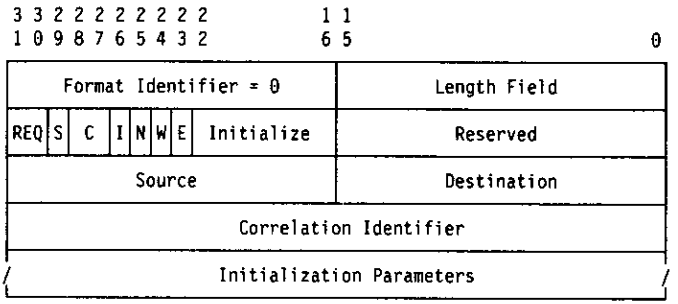

Figure 1-26. Initialization Request Control Element

The initialization parameters passed to the server in the Entity-to-Entity field of the request control element are server dependent and need to be defined on a server-by-server basis.

If the indirect bit is set in the Common Indicators field of the initialization request control element, then the Entity-to-Entity field does not contain the actual initialization parameters. Instead, it contains the location and the length of a data area in shared memory where they are stored.

A server reports the successful completion of an initialization request by returning a reply control element to the requestor. The Source, Destination, and Correlation Identifier fields from the initialization request control element, along with any return values, are returned to the requestor in the reply control element.

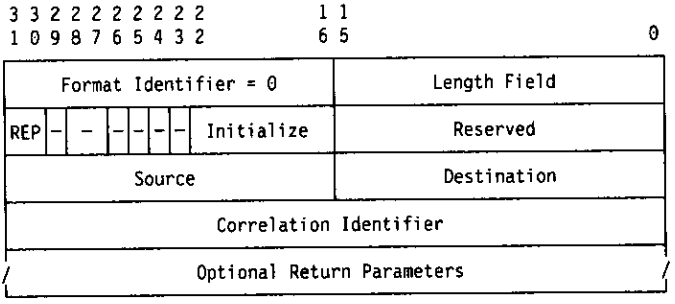

Figure 1-27. Initialization Reply Control Element

The optional return parameters passed in the Entity-to-Entity field of a reply control element are server dependent.

A client can choose to suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the initialization request control element.

If unsuccessful, an error control element containing the Source, Destination, and Correlation Identifier fields from the initialization request control element, along with status information identifying the cause of the failure, are returned to the requestor.

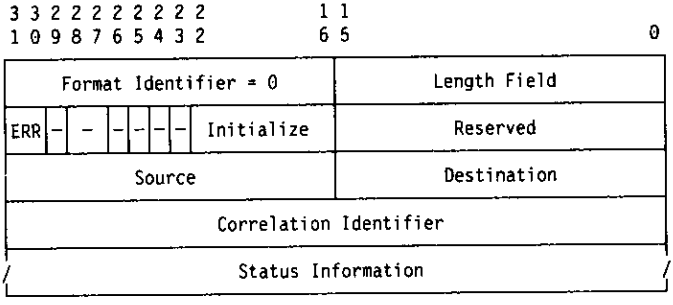

Figure 1-28. Initialization Error Control Element

Read

The Read function is used in request, reply, and error control elements. In a request control element, it is used to set up and initiate the transfer of data from a server to a client. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field containing the parameters required by the server to perform the read, are passed to a server in the request control element.

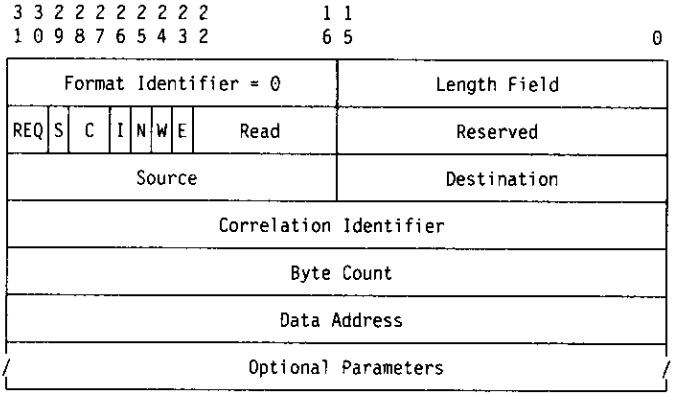

Figure 1-29. Read Request Control Element

The parameters contained in the Entity-to-Entity field of a request control element are not limited to those identified above. Additional parameters can be supplied to meet the requirements of each client and server relationship.

If the indirect bit is set in the Common Indicators field of the read request control element, the Entity-to-Entity field does not contain the byte count, data address, and optional parameters. Instead, it contains the location and the length of a data area in shared memory where these parameters are stored.

When a server has successfully completed the transfer, it returns a reply control element with the Source, Destination, and Correlation Identifier fields from the read request control element and the actual number of bytes transferred during the read operation.

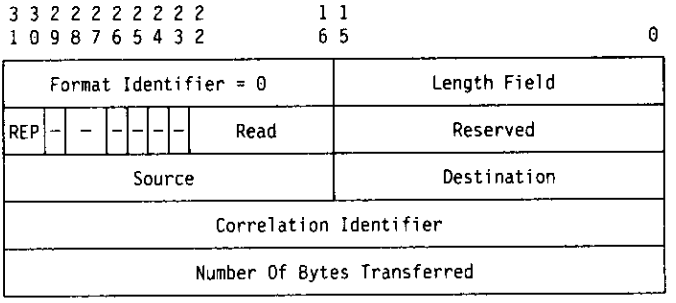

Figure 1-30. Read Reply Control Element

A client can choose to suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the read request control element.

If unsuccessful, an error control element containing the Source, Destination, and Correlation Identifier from the read request control element, along with status information identifying the cause of the failure, is returned to the requestor.

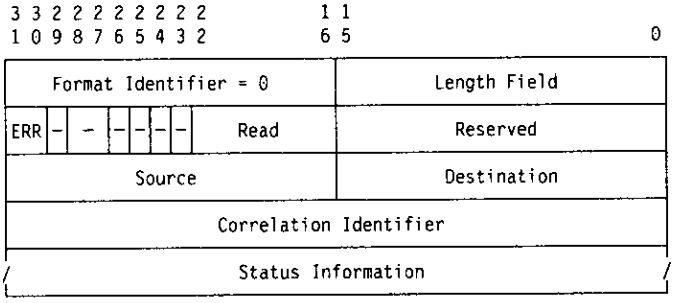

Figure 1-31. Read Error Control Element

Read List

The Read List function is used in request, reply, and error control elements. It is used to establish and initiate the transfer of data and control information from a server into several, possibly noncontiguous, areas in shared memory. This is often referred to as data chaining. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field containing the list of byte counts and data addresses required by the Read List function, are passed to the server in the request control element.

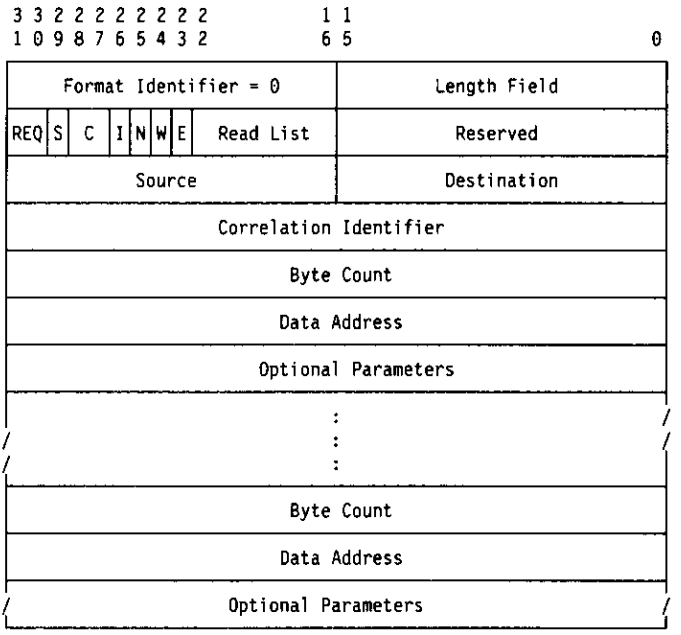

Figure 1-32. Read List Request Control Element

If the Indirect list is indicated in the Common Indicators field, the Entity-to-Entity field does not contain the byte count, data address, or optional parameters. Instead, it contains the location and the length of an area in shared memory where the actual list is stored.

When a server has successfully completed the transfer, it returns a reply control element with the Source, Destination, and Correlation Identifier fields from the read list request control element, and residual-byte count in the reply control element. The format and content of the read list reply control element is the same as the read reply control element (see "Read" on page 1-42).

A client can choose to suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the read list request control element.

If unsuccessful, the following are returned to the requester in an error control element.

- The Source, Destination, and Correlation Identifier fields from the read list request control element

- The status information identifying the cause of the failure
- The pointer to the last data address that was processed
- The actual number of bytes read.

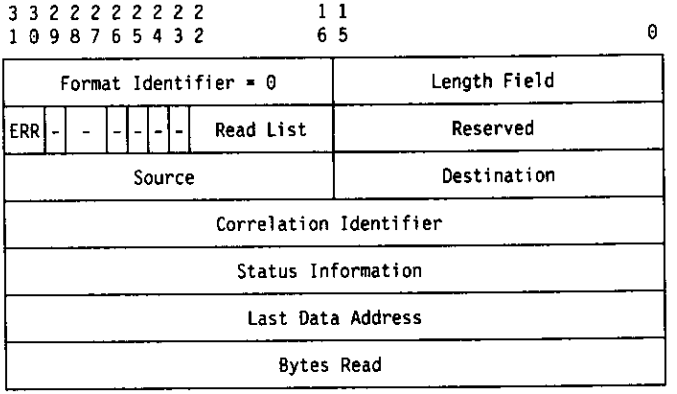

Figure 1-33. Read List Error Control Element

Read Immediate

The Read Immediate function is used in request, reply, and error control elements. It is used to request the transfer of data and control information from a server to a client. It differs from the Read Request function in that the data is to be returned in the Entity-to-Entity field of the reply control element where it becomes immediately available to the client. The Source, Destination, Correlation Identifier fields, and the Entity-to-Entity field, containing the parameters required by the Read Immediate function, are all passed to a server in the request control element.

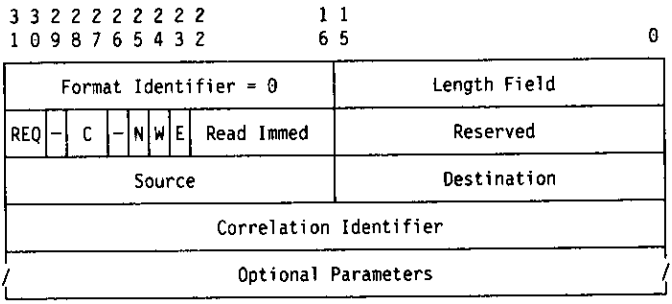

Figure 1-34. Read Immediate Request Control Element

Note: The amount of data that can be returned using the read immediate request control element is configuration dependent. That is, it is directly related to the size of the delivery pipe (queue). Therefore, care should be exercised when using this request control element.

To complete the requested operation, the server returns a reply control element with the Source, Destination, and Correlation Identifier fields from the read immediate request control element. In the Entity-to-Entity field, it returns the immediate data.

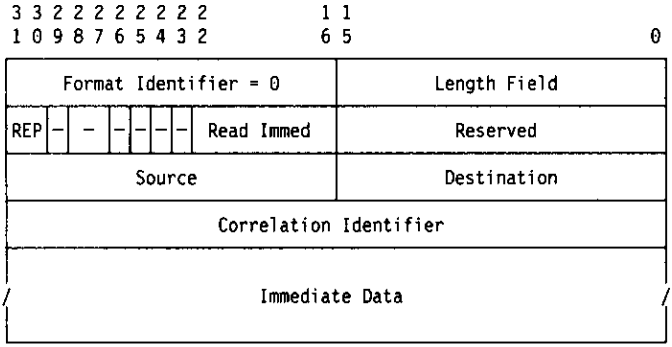

Figure 1-35. Read Immediate Reply Control Element

The Read Immediate function ignores the suppress bit if it is set in the Common Indicators field of the read immediate request control element because it always returns a reply control element.

If unsuccessful, an error control element containing the Source, Destination, Correlation Identifier fields from the read immediate request control element, and status information identifying the cause of the failure, are returned to the requestor.

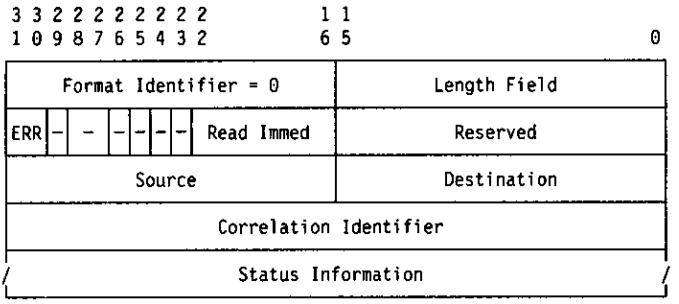

Figure 1-36. Read Immediate Error Control Element

Write

The Write function is used in request, reply, and error control elements. It is used to set up and initiate the transfer of data and control information from a client to a server. The Source, Destination, Correlation Identifier, and the Entity-to-Entity fields are all passed to a server in the request control element.

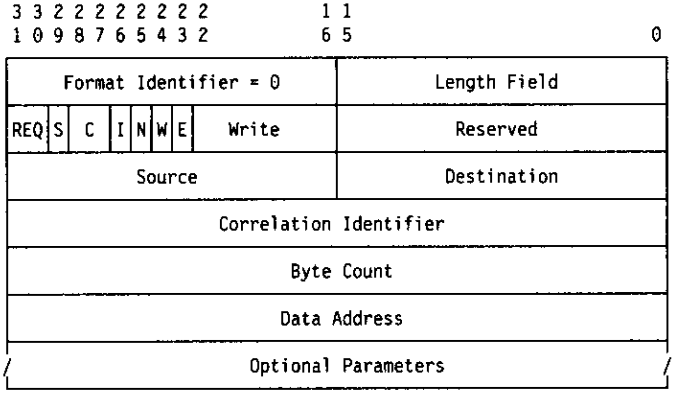

Figure 1-37. Write Request Control Element

The Entity-to-Entity field contains the amount and location of the data to be transferred by the Write function. The Entity-to-Entity field is not limited to these parameters. Additional parameters can be supplied to meet the requirements of each client and server relationship.

If the indirect bit is set in the Common Indicators field of the write request control element, the Entity-to-Entity field does not contain the byte count, data address, or optional parameters. Instead, it contains

the location and length of the area in shared memory where data is stored.

When the server completes the transfer, it returns a reply control element with the Source, Destination, and Correlation Identifier fields from the write request control element.

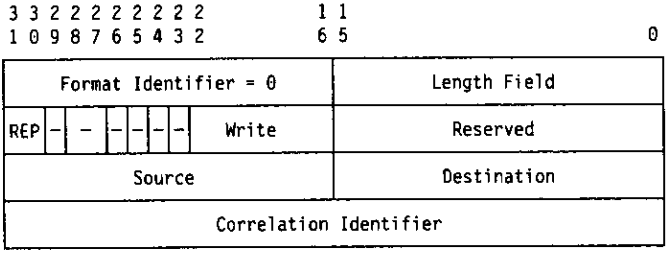

Figure 1-38. Write Reply Control Element

A client can suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the write request control element.

If unsuccessful, an error control element containing the Source, Destination, and Correlation Identifier fields from the write request control element, and status information identifying the cause of the failure, is returned to the requestor.

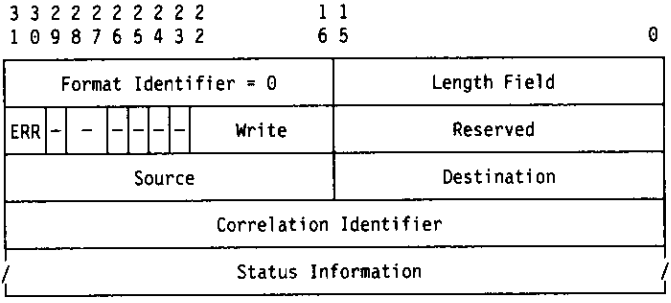

Figure 1-39. Write Error Control Element

Write List

The Write List function is used in request, reply, and error control elements. It is used to set up and initiate the transfer of data and control information to a server from several possibly noncontiguous areas of shared memory. This is often referred to as data chaining. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field containing the list of byte counts, data addresses, and optional parameters required by the Write List function, are all passed to the server in the request control element.

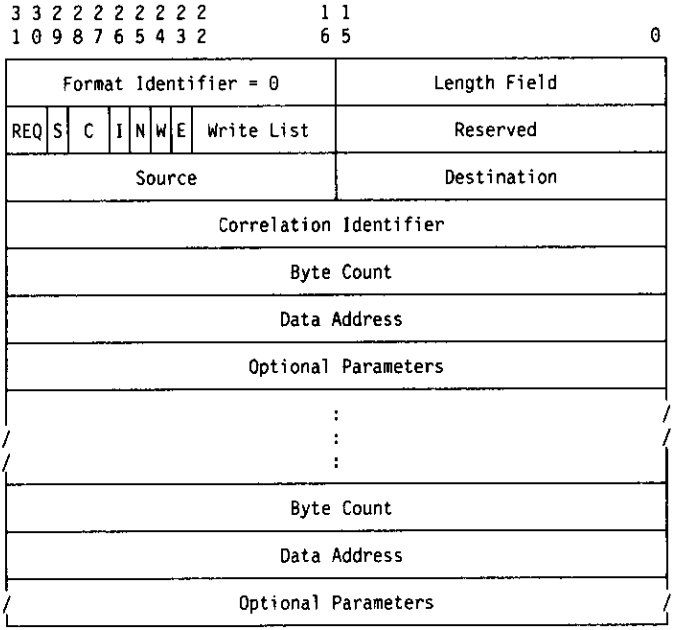

Figure 1-40. Write List Request Control Element

If the indirect bit is set in the Common Indicators field of the write list request control element, the Entity-to-Entity field does not contain the byte count, data address, and optional parameters. Instead, it contains the location and length of the area in shared memory for the list.

When the server has completed the transfer, it returns a reply control element with the Source, Destination, Correlation Identifier fields from the write list request control element, and residual-byte count in the Entity-to-Entity field of the reply control element. The format and

content of the write list reply control element is the same as the write reply control element (see "Write" on page 1-47).

A client can choose to suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the write list request control element.

If unsuccessful, the following are returned to the requester in an error control element:

- The Source, Destination, and Correlation Identifier fields from the write list request control element
- The status information identifying the cause of the failure
- The pointer to the last data address that was processed
- The actual number of bytes transferred.

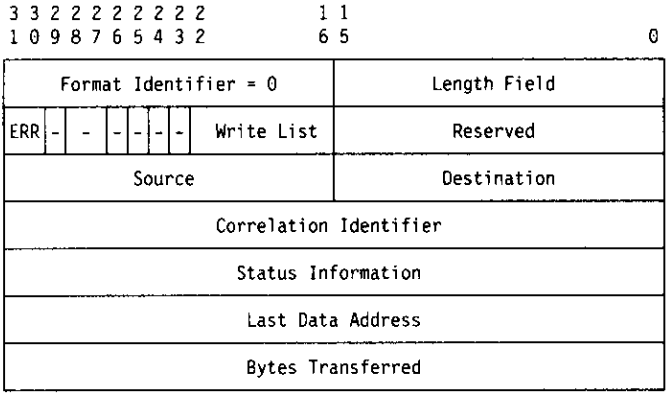

Figure 1-41. Write List Error Control Element

Write Immediate

The Write Immediate function is used in request, reply, and error control elements. It is used to carry data and control information from a client to a server. It differs from the write request in that the data is present in the Entity-to-Entity field of the request control element and is immediately available to the server. The Source, Destination, and Correlation Identifier fields, and data are passed to a server in the request control element.

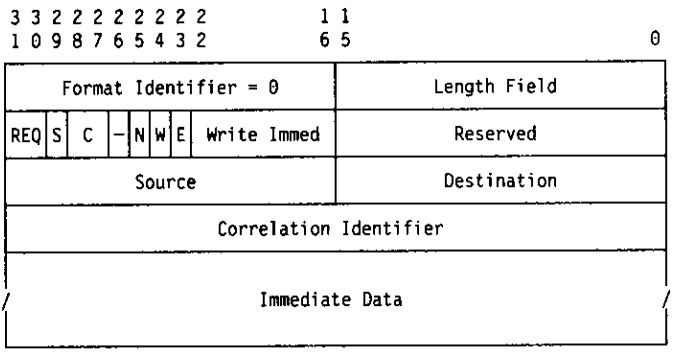

Figure 1-42. Write Immediate Request Control Element

The format and meaning of the data contained in the Entity-to-Entity field of the request control element are client and server dependent and, as such, are not defined here.

Note: The amount of data that can be transferred using the write immediate request control element is configuration dependent. That is, it is directly related to the size of the delivery pipe (queue). Therefore, care should be exercised when using this request control element.

When the server has successfully received the data, it returns a reply control element with the Source, Destination, and Correlation Identifier fields from the write immediate request control element and, in the Entity-to-Entity field, a count of the actual number of bytes received.

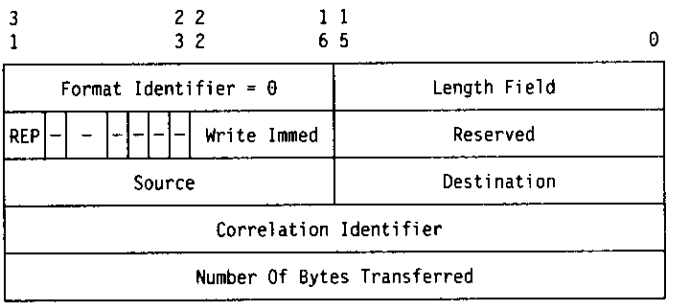

Figure 1-43. Write Immediate Reply Control Element

A client can suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the write immediate request control element.

If unsuccessful, an error control element is returned to the requestor. This control element contains the Source, Destination, and Correlation Identifier fields from the write immediate request control element, and status information identifying the cause of the failure.

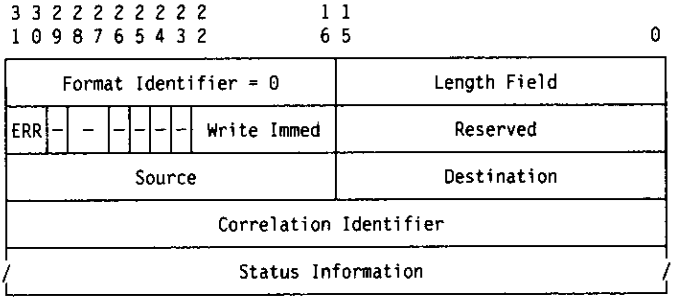

Figure 1-44. Write Immediate Error Control Element

Execute List

The Execute List function is used in request and reply control elements. It is used to set up a repetitive loop for executing a list of request control elements one or more times. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field containing the location, length, and repetition count required by the Execute List function are all passed to a server in the request control element.

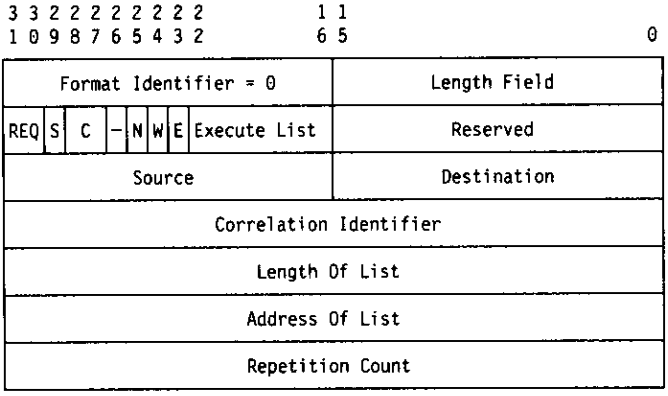

Figure 1-45. Execute List Request Control Element

The list can consist of one or more request control elements that must be contiguous in shared memory. Because each control element in the list contains its own length field, the start of the next control element can be determined. The Length of List field determines where the list ends. On each pass through the list, the repetition count is decremented by 1, and if not 0, the list is executed again. If the initial repetition count is negative or 0, the request terminates.

The request control elements contained in the list can use any available function code and common indicator.

When the repetition count goes to 0, the list is terminated and the server returns a reply control element with the Source, Destination, and Correlation Identifier fields from the original execute list request control element.

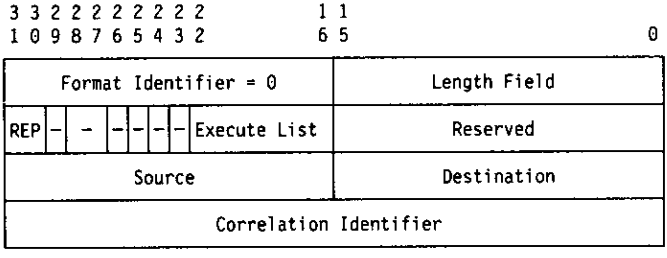

Figure 1-46. Execute List Reply Control Element

A client can choose to suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the execute list request control element.

If an error occurs while processing any control element within the list, the list is terminated and the control element with the error is returned.

Mark

The Mark function is used only in a request control element to provide a means of synchronization between a client and a server. Its only function is to pass the Correlation Identifier field to the server.

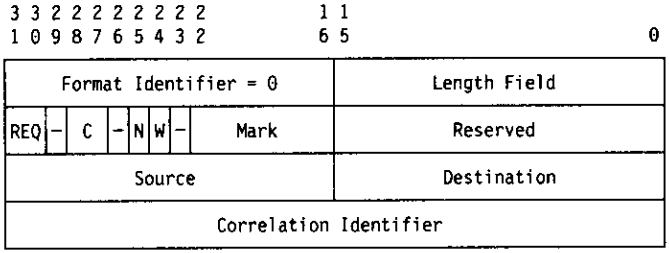

Figure 1-47. Mark Request Control Element

The Mark function can utilize the notify and wait bits to cause a server to inform a client of the current processing state of the server. For more details refer to "Notification and Wait" on page 4-11.

The Mark function ignores the indirect, suppress, and expedite bits if set in a mark request control element.

Cancel

The Cancel function is used in the request, reply, and error control elements. In a request control element, it is used to cancel one or more outstanding requests. The Source, Destination, Correlation Identifier, and Entity-to-Entity fields contain the cancellation qualifier and the cancellation list of request control element correlation identifiers.

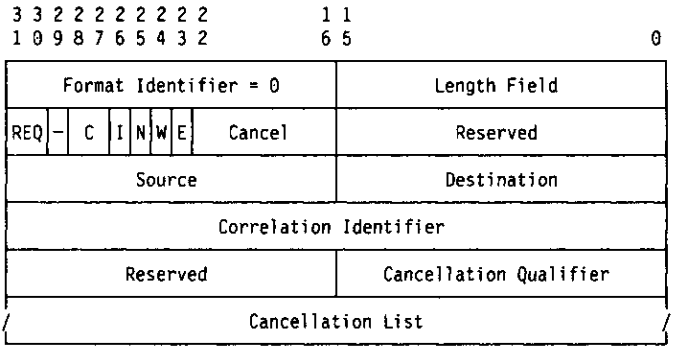

Figure 1-48. Cancel Request Control Element

The cancellation qualifier bits identify the request control elements to be cancelled:

- All outstanding requests (bit 0 = 1)
- Requests that match the Source ID (bit 1 = 1)
- First request matching the cancellation list value (bit 2 = 1)
- All requests matching the cancellation list value (bit 3 = 1).

If bits 2 or 3 of the cancellation qualifier are set to 1, a cancellation list is present. If bit 1 is set, then bit 2 or 3 must also be set.

If the indirect bit is set in the Common Indicators field of the cancel request control element, the Entity-to-Entity field does not contain the cancellation list. Instead, it contains the location and length of the area in shared memory for the list.

When the server has successfully completed the Cancel function, it returns a reply control element to the requestor. The Source, Destination, and Correlation Identifier fields from the cancel request control element are returned in the reply control element.

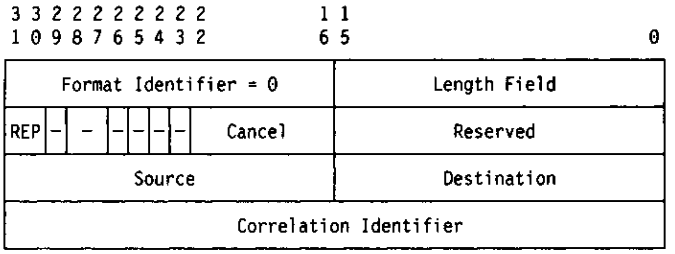

Figure 1-49. Cancel Reply Control Element

The server tries to cancel all the requests whose Correlation Identifier fields are in the Entity-to-Entity field of the cancel request. If the server cannot cancel all requests, it returns an error control element with the status and the list of Correlation Identifiers fields from the cancelled requests. An empty list indicates that it was unable to perform the cancel on any of the identified control elements.

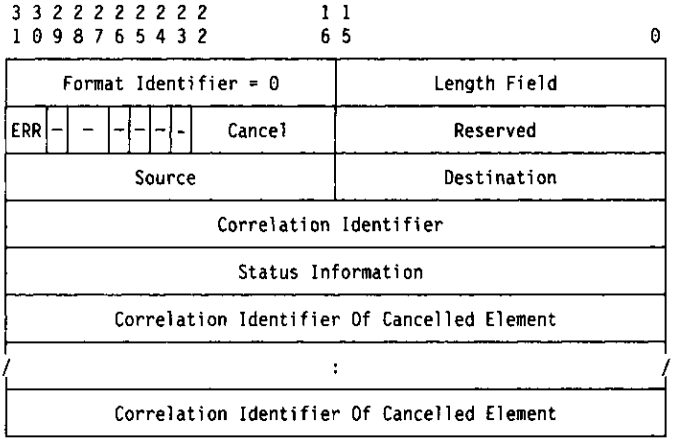

Figure 1-50. Cancel Error Control Element

However, a cancel request control element with the *all outstanding control element* qualifier set does not return an error control element with the list of cancelled control elements.

Reset

The Reset function is used in request, reply, and error control elements. In a request control element, it is used to request a server to place itself into a known state. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field containing the desired state, are all passed to a server in the request control element.

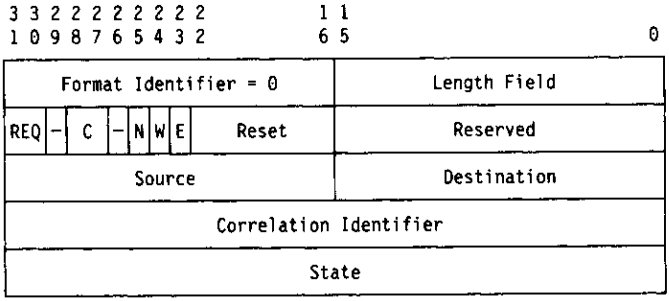

Figure 1-51. Reset Request Control Element

A reset request control element with a value of 0 for the State field requests the device to return to the initialization state. All other values are implementation dependent.

When reset to the initialization state, the server purges all work in all queues, and any outstanding requests are terminated with an error control element. All requests except for the initialize control element are rejected with an error control element when in the reset-to-initialization state.

When the server has completed the reset, it returns a reply control element with the Source, Destination, and Correlation Identifier fields from the request control element to the client.

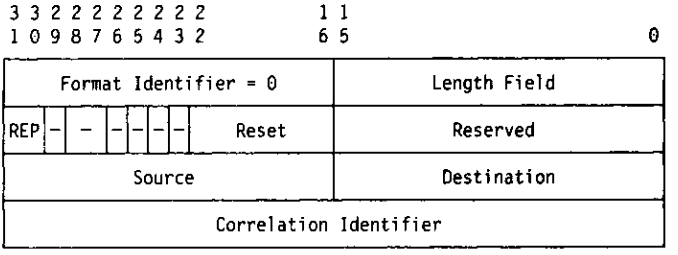

Figure 1-52. Reset Reply Control Element

The Reset function ignores the suppress and indirect bits, if the bits are set in a reset request control element.

If unsuccessful, an error control element containing the Source, Destination, and Correlation Identifier fields from the reset request control element, and status information identifying the cause of the failure, are returned to the requestor.

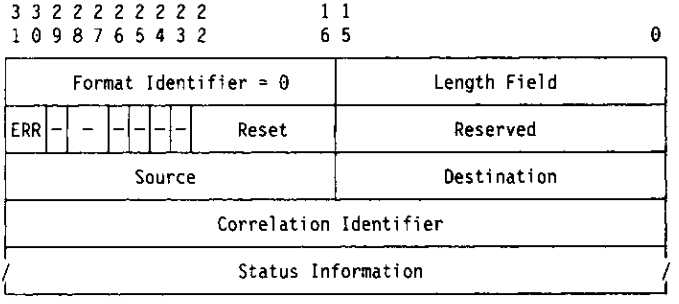

Figure 1-53. Reset Error Control Element

Read Configuration

The Read Configuration function is used in request, reply, and error control elements. When used in a request, it causes the server to return configuration information to the client. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field containing the byte count, data address, and optional parameters are passed to the server in the request control element.

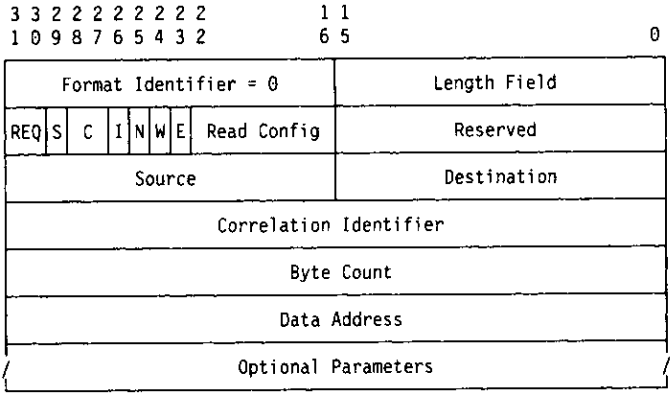

Figure 1-54. Read Configuration Request Control Element

The parameters contained in the Entity-to-Entity field of a read configuration request control element are not limited to those identified above. Additional parameters can be supplied to meet the requirements of each client and server relationship.

If the indirect bit is set to 1 in the Common Indicators field of the read configuration request control element, the Entity-to-Entity field does not contain the byte count, data address, and optional parameters. Instead, it contains the location and the length of the data area in shared memory where the parameters are stored.

A server reports the successful completion of a read configuration request by returning a reply control element to the requestor. The Source, Destination, and Correlation Identifier fields are returned in the reply control element.

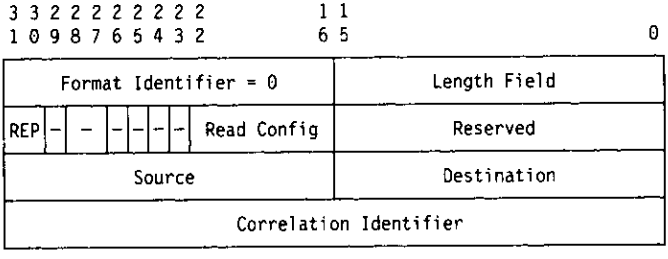

Figure 1-55. Read Configuration Reply Control Element

A client can suppress returning a reply control element by setting the suppress bit to 1 in the Common Indicators field of the read configuration request control element.

If the server is unable to report configuration information, an error control element is returned to the requestor. This control element contains the Source, Destination, and Correlation Identifier fields from the read configuration request control element, along with the reason.

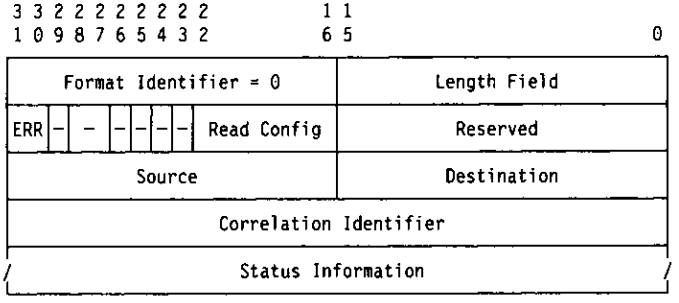

Figure 1-56. Read Configuration Error Control Element

Diagnose

The Diagnose function is used in request, reply, and error control elements. In a request, the Diagnose function initiates diagnostic routines by a server. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field which identifies the specific diagnostic tests to be run, as well as any operational parameters, are passed to the server in the request control element.

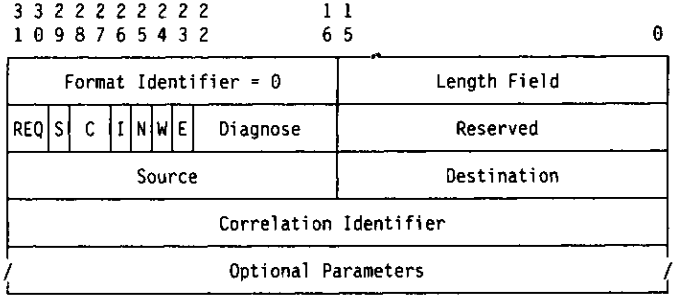

Figure 1-57. Diagnose Request Control Element

If the indirect bit is set in the Common Indicators field of the diagnose request control element, the Entity-to-Entity field does not contain the optional parameters. Instead, it contains the location and length of the area in shared memory for the parameters.

A server reports the results of running the diagnostic tests by returning a reply control element to the requestor. The Source, Destination, and Correlation Identifier fields from the diagnose request control element, and the Entity-to-Entity field containing the test results, are returned to the reply control element.

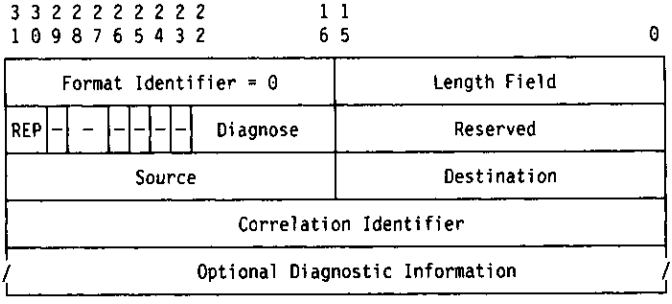

Figure 1-58. Diagnose Reply Control Element

A client can suppress returning a reply control element by setting the suppress bit in the Common Indicators field of the diagnose request control element.

If a server is unable to perform the requested diagnostic test, an error control element is returned and the Entity-to-Entity field indicates the cause of the error. This error control element contains the Source, Destination, and Correlation Identifier fields from the diagnose request control element.

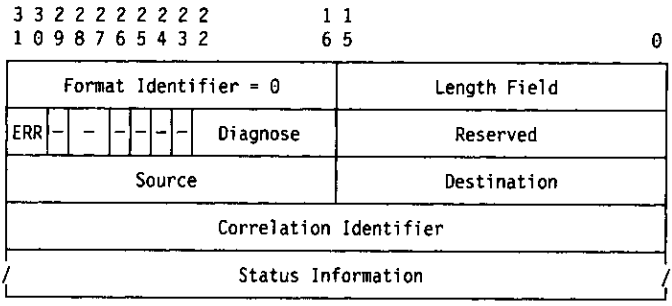

Figure 1-59. Diagnose Error Control Element

Event Control Elements

The event control element is provided by the delivery service in addition to the request, reply, and error control elements. It provides information about the progress of a request, or about the side-effects of a previous request. The event control element differs from the other control elements in that it reports a change in the state of one of the communicating entities (client or server). Because of the impact of these state changes, the entity must be informed of the change as quickly as possible. The following sections describe the format, content, and use of event control elements.

Resume

A resume event is used to notify a server of a change in a clients processing state. A client uses this event to notify a server that a client is now in a state that allows the processing of request control elements to be resumed. The Source, Destination, and Correlation Identifier fields of the original request control element causing the suspension are returned to the server in the resume event control element.

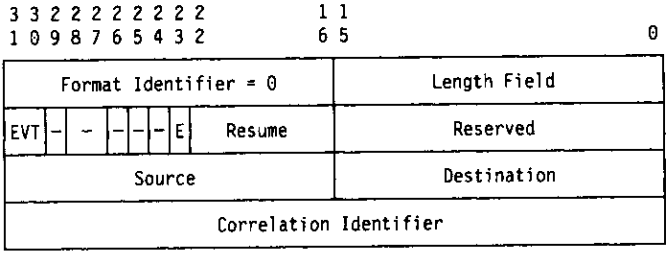

Figure 1-60. Resume Event Control Element

A resume event is ignored if it is received from a client other than the one responsible for the suspended state or from a client that is not in the suspended state.

Notification

A notification event is used to confirm the receipt of a request control element with the notification bit of the Common Indicators field set to 1 (see "Notification and Wait" on page 4-11). The Source, Destination, and Correlation Identifier fields of the request control element containing the notification bit, are returned to the client in the Entity-to-Entity field of the event control element.

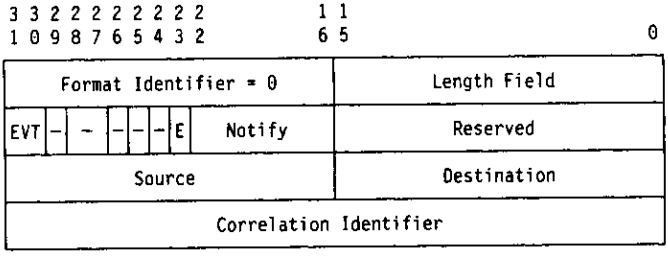

Figure 1-61. Notification Event Control Element

Inform

An inform event is used to provide a means of conveying information regarding the processing state of either a client or a server from one to the other.

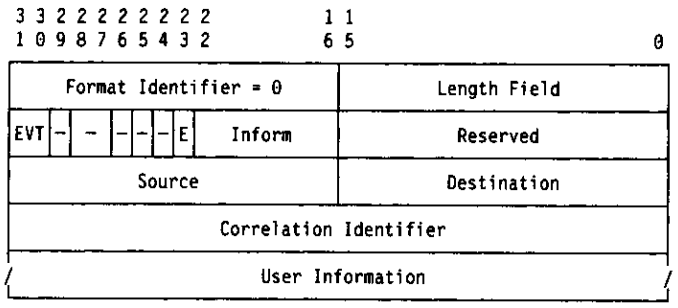

Figure 1-62. Inform Event Control Element

Wrap

A wrap event is a management and control-initiated event used internally by the delivery service. It is used to synchronize the operations associated with the in-bound or out-bound delivery pipes.

A wrap event control element can be sent only by the delivery service. This control element informs the delivery service that there was insufficient space in the pipe for the next control element. Therefore, the control element is placed in the start of the pipe.

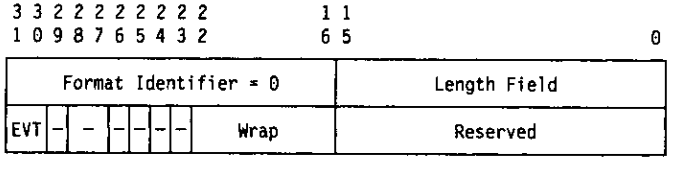

Figure 1-63. Wrap Event Control Element

Chapter 2. Physical Level

At the physical level, Move mode defines the services, functions, and protocols that are used by the delivery-support logic to physically manage control-element delivery between units attached to the channel.

The physical level provides a set of services for gaining access to I/O and memory address space, and for signalling between units that are attached to the channel that is independent of the hardware. This allows different forms of system unit and adapter hardware to be used by the same delivery-support logic.

The physical level allows for differences in the way services are provided by various system units and adapters. The location and use of control areas; the method of generating and handling interrupts; and the protocols used to access I/O and memory address space are transparent to the logic at the delivery level. Only the physical level needs to be aware of how the services are mapped to the underlying hardware.

Physical-level services can be used to push or pull data between local memory address space and control areas in shared memory. The data can be signalling information, surrogate-state information, a control element, or data pointed to by the control elements.

Structure

The following figure shows the physical-level structure for a configuration with a single system unit (Unit X) and two adapters (Unit Y and Unit Z). The services of the physical level are provided by a combination of hardware, physical-support logic in each unit, control areas in I/O address space, and control areas (signalling) in shared memory.

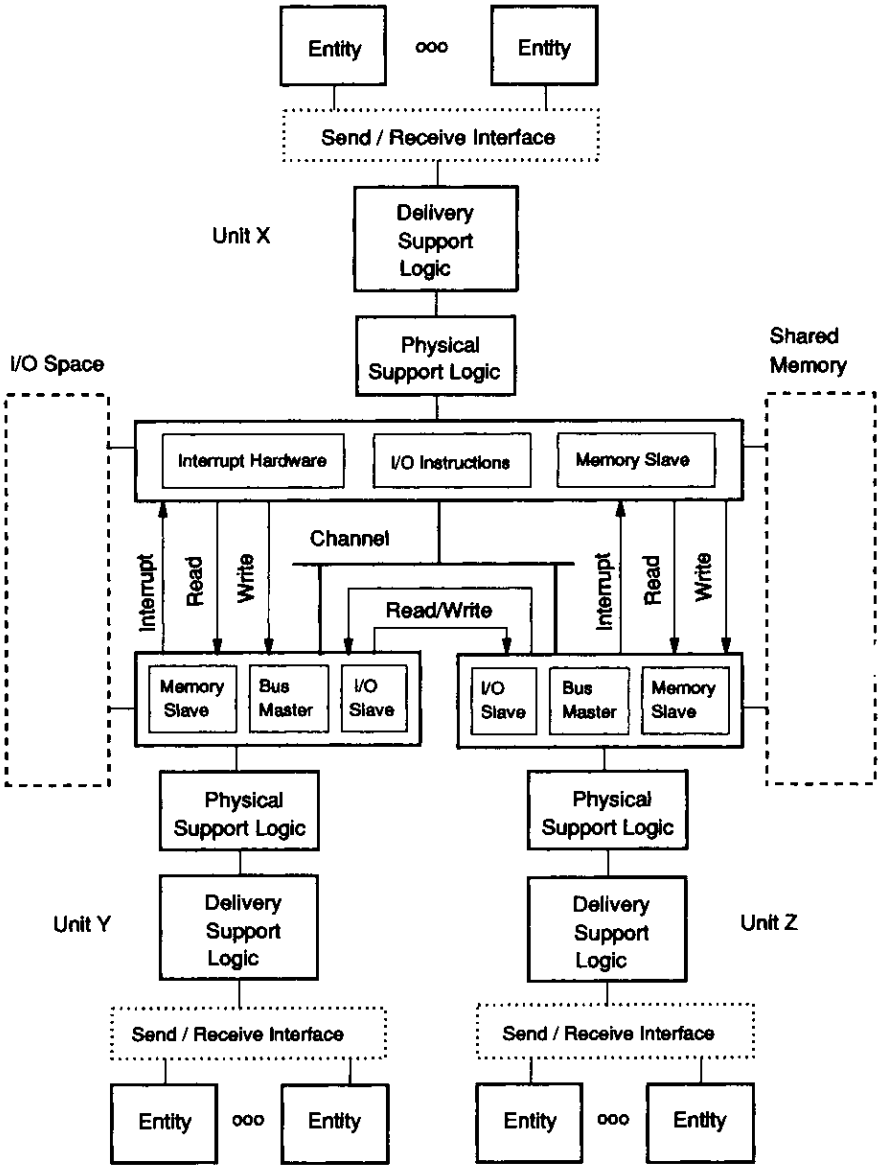

Figure 2-1. Physical Level

System

System unit designs typically provide support for system interrupts, memory slaves, and I/O slaves.

The interrupt controller provides hardware support for signalling system interrupts from an adapter to a system unit. The memory-access instructions (such as Load and Store) provide a means for gaining access to data and control information located in shared memory. The I/O-access instructions (such as In and Out) provide a means for gaining access to control information located in control areas in I/O address space. The bus-master support allows adapters to control the channel.

Adapter

Adapter designs typically provide support for I/O slaves, bus masters, and optional memory slaves. The I/O slave provides hardware support for initializing an adapter, enabling access to an adapter, signalling between adapters, and identifying the adapter that is the source of an interrupt. The memory slave provides hardware support for gaining access to data and control areas in shared memory from a system unit or from another adapter with bus-master capabilities. The bus master provides hardware support for gaining access to data and control areas in either shared or local memory and control areas in I/O address space in other system units or adapters.

Support Logic

The support logic is responsible for implementing the set of services provided to the delivery level for the specific hardware used by an adapter or system unit. The protocols used between two units are determined by the hardware and must be implemented by the support logic in the source and destination units. Support logic must be provided for the following services: push, pull, and signal.

Push and Pull

The support logic for the push and pull service must establish the addresses and the method to be used to move the data into or out of shared memory. The protocols can vary for different combinations of system units and adapters. The following sequence is used to establish the addressing for performing the Push or Pull functions.

- During system configuration, the specific I/O and shared memory address ranges are defined for the various units.
- During initialization, specific control areas in shared memory are established. Initialization includes establishing any address translations between the channel address form and the local memory address form.
- Adapters can establish the addresses just before performing the push or pull operation. In this case, establishing the address can be done locally or it can be visible on the channel.

The data can be moved in one of several ways, depending on the functions available in the adapter. The data migration includes hardware to provide an interface from the unit to the channel, as well as providing the following functions.

- If a unit (such as a system) has hardware that supports memory-mapped I/O instructions, these instructions are used to push and pull data.
- If a unit (such as an adapter) has bus-master capabilities, it is used to push and pull data between units.

The hardware in each unit that is used to support the push and pull operations can also be shared with the protocols that move data that is pointed to by the control elements.

The architecture requires control areas in memory to be aligned on word (2-byte) boundaries so that they are accessed as words in shared memory. This eliminates the possibility of having word-aligned surrogate information split when it is accessed by the delivery-support logic.

Signal

The context for describing the signalling logic is shown in Figure 2-2 on page 2-6. The signalling service is supported by a combination of hardware and support logic in the sending and receiving units. The support logic in the receiving unit uses the information in the Signalling Control areas to determine which units have interrupts pending and the reasons for those interrupts.

The send-signal logic and receive-signal logic operate in support of the signalling protocols defined at the delivery level.

- The send-signal logic is used by the delivery or management-support logic to cause the interrupt of another unit after it has pushed signalling information into the Signalling Control area assigned to it in the other unit.
- The receive-signal logic examines all Signalling Control areas whenever a hardware interrupt is received. This ensures that all signalling indications are captured even though an interrupt could have been missed by a hardware-busy or masked condition.

  Using the information in the Signalling Control areas, the receive-signal logic posts the appropriate delivery-support logic at the delivery level and clears the contents of the Signalling Control area to 0. For management, it posts the unit-management logic and resets the management indicator in the appropriate Signalling Control area.

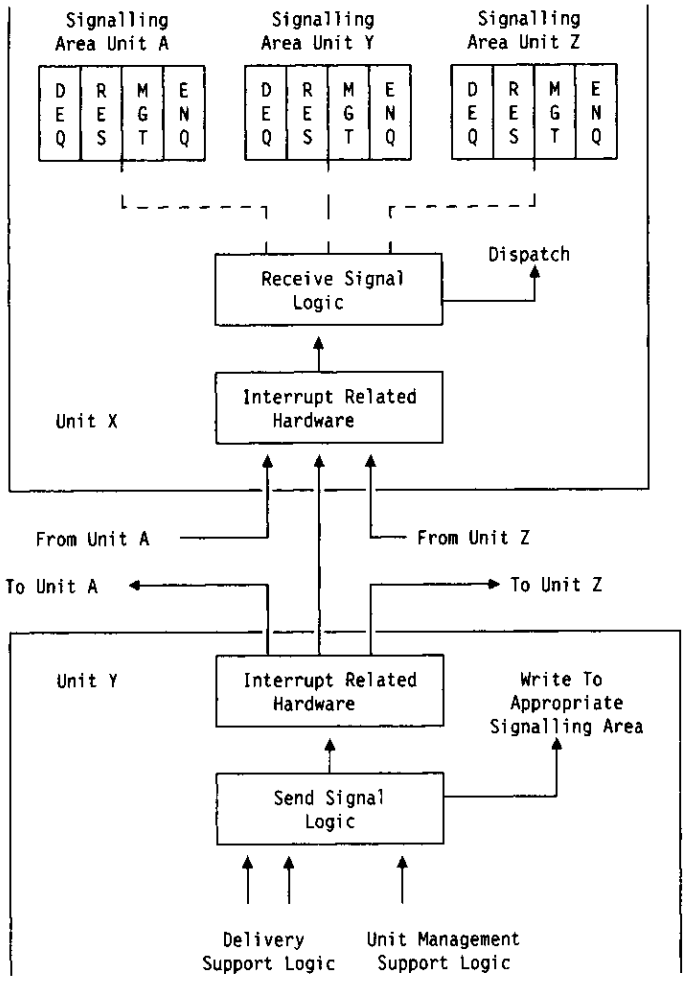

Figure 2-2. Signalling

The signalling service operates in a variety of physical implementations by using Signalling Control areas in shared memory and simple signalling primitives. To support peer-to-peer delivery, each sending unit has a separate Signalling Control area that identifies it to the receiving unit. An indication in a particular Signalling Control area defines which unit is signalling. The receiving unit has a Signalling Control area for each unit it communicates with.

The structure and content of the Signalling Control area is described in detail in "Signalling Control Area" on page 1-29.

The support logic for the signalling portion can be viewed as having a part that establishes access to the interrupt hardware and a part that causes an interrupt in the destination unit.

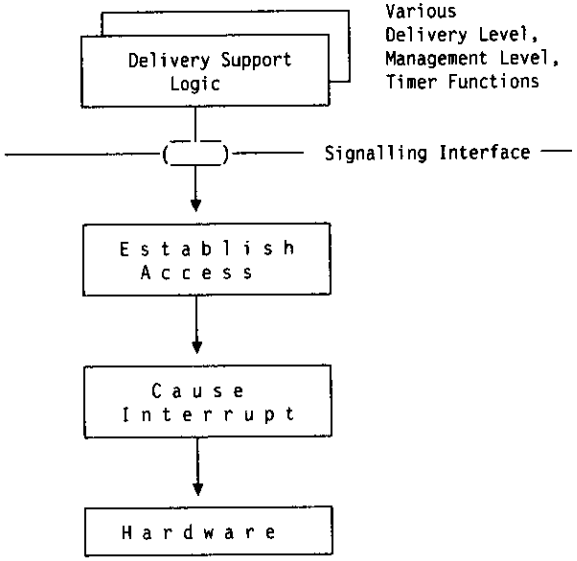

Figure 2-3. Interrupt Support Logic

The protocols can vary for differences:

- Between unit types (adapters and systems)
- In types of adapters (such as LAN, COMM, and SCSI)
- In the processor architectures used for the specific hardware.

Access to the interrupt hardware is established at specific times:

- During system configuration, the interrupt level is allocated along with the I/O addresses used by the adapter.
- During adapter initialization, the locations of the specific control areas, including the Signalling Control area, are established. The type of unit determines the location of the control area and the value that causes an interrupt.

- During operation, when the interrupt is shared, then some form of interrupt-sharing protocol could be required.

The specific protocol for generating the interrupt is determined by the type of unit being signalled.

- If the unit is a system unit, an interrupt is caused by hardware (IRQ) lines in the channel that connect to the interrupt controller hardware.
- If the unit is an adapter, an interrupt is caused by writing a predefined value into a specific control area (port) in I/O address space. The value and location are specified as part of the configuration and initialization process.

Memory Address Space

The physical level provides access to data and control areas in shared memory.

The physical-level and delivery-level protocols in the source and destination units must have access to (and in some instances share) data and control information in the same address space.

At the delivery level, this includes the control elements in the data areas associated with each delivery pipe, as well as their associated Surrogate Control areas.

At the physical level, this includes the Signalling Control areas associated with each unit (system and adapter). The information in the Signalling Control area is also accessed by the support logic implementing the delivery-level protocol in the source and destination units.

Assigning these control areas to locations in shared memory allows them to be accessed, and the information they contain to be shared, by different units on the channel.

The data buffers used to support certain forms of entity-to-entity data transfers (defined by addresses within specific request and reply control elements) are also assigned locations in shared memory. The location of these data buffers is determined by the logic at the

processing level, which supports entity-to-entity data transfers, and can change from control element to control element.

The shared memory structure utilizes the physical addresses on the channel to define the address space that is shared between system units and adapters.

The control areas in shared memory are used for the delivery pipes and their associated Surrogate Control areas. The location of each control area is assigned either at system configuration or adapter initialization time and remains constant throughout the operation of the system. The shared memory can be physically located in either the system unit, the adapter, or both. Each of these areas is accessed using the memory-slave support in their respective units.

The physical-support logic handles the access to all control information in the various shared memory locations using the appropriate hardware for the specific unit (system unit or adapter) performing the access.

Note: The ability to assign shared memory to various units and the ability to assign control areas to any location in shared memory allows for various cost-performance tradeoffs to be implemented using the same delivery-level and processing-level protocols.

I/O Address Space

The physical-support logic provides access to a set of control areas (ports) in I/O address space that are confined to, and shared at, the physical level between system units and adapters. This space is shared across units and is supported by the I/O slave support in adapters.

The range of I/O addresses assigned to each adapter is defined by programmable options select (POS) parameters provided during system setup. The location and use of control areas in this I/O address space for a given adapter type can be found in "I/O Address Space" on page 1-15.

Physical Level Services

Operations provided by the physical level are used to access control information in locations in shared memory or I/O address space and to support the sending and receiving of signals by any system unit or adapter on the channel. The operations provided are:

- Push
- Pull
- Signal.

The delivery-support logic uses the I/O address space to move control parameters between the system unit and adapter. It uses the shared memory to move state information and variable-length control elements between delivery-support logic in different units (system unit as well as adapters).

A system unit can use processor memory-mapped I/O or programmed I/O instructions to access control areas in I/O address space.

In an adapter, these operations can be provided by memory-mapped instructions or by bus-master operations, depending upon the specific system unit or adapter implementation.

Push

The push operation moves control information from a location in the caller's memory to a location in shared memory.

Note: The caller's memory can be located in local address space or in shared memory address space.

The push operation is an internal function and is not defined by the architecture. However, the following conceptual service primitive illustrates the parameters required when invoking the service:

```
PUSH (unit id, address space, from address, to address, length, return code)
```

Unit ID This parameter identifies the system unit or adapter that is the target of the push operation.

Address space This parameter indicates whether the to address argument refers to a location in shared memory or I/O address space.

From address This parameter identifies the location in the caller's memory that is the source of the information to be pushed into the location in shared memory or I/O address space indicated by the to address parameter.

To address This parameter identifies the location in shared memory or I/O address space that is to be the destination for the information indicated by the from address parameter.

Length This parameter indicates the number of bytes to be pushed into the location in shared memory or I/O address space identified by the to address parameter.

Return code This parameter indicates the success or failure of the push operation.

Pull

The pull operation moves information from a location in I/O or shared memory to a location in the caller's memory.

Note: The caller's memory can be located in local address space or shared memory address space.

The pull operation is an internal function and is not defined by the architecture. However, the following conceptual service primitive illustrates the parameters required when invoking the service:

```
PULL (unit id, address space, from address, to address, length, return code)
```

Unit ID This parameter identifies the system unit or adapter that is the target of the pull operation.

Address space This parameter indicates whether the from address argument refers to a location in shared memory or I/O address space.

From address This parameter identifies the location in I/O space or shared memory that is the source of the information to be pulled into the caller's memory as indicated by the to address parameter.

To address This parameter identifies the location in the caller's memory that is the destination for the information indicated by the from address parameter.

Length This parameter indicates the number of bytes to be pulled from the location in I/O or shared memory identified by the from address parameter.

Return code This parameter indicates the success or failure of the pull operation.

Signal

In addition to the push and pull operation primitives, the physical level also provides a service primitive for signalling between units. The physical-level logic at the source unit maps the signalling primitive into the appropriate set of hardware signals to cause an interrupt at the destination unit (system unit or adapter).

The signal operation is an internal function and is not defined by the architecture. However, the following conceptual service primitive illustrates the parameters required when invoking the service.

```
SIGNAL (unit id, area, value, return code)
```

Unit ID This parameter identifies the system unit or adapter.

Area This parameter identifies the specific area within the Signalling Control area (the enqueue area, the dequeue area, or the management area).

Value This parameter identifies the value to be written to the specified area.

Return code This parameter indicates the success or failure of the signal operation.

Data Delivery

In adapters, bus-master support can also be used to transfer data to or from entities in the adapter. The request and reply control elements contain pointers to the location in shared memory of the data to be accessed by the adapter entities. The push and pull operations can be used by the entities at the processing level, and the support-logic at the delivery level, to access data in shared memory.

Physical Level Protocols

This section defines the architecture for the various types of physical-level protocols supported. The definition for each type includes the protocols for both system unit to adapter and for adapter to adapter.

Control Areas

The physical level uses the following parameters that are part of the system configuration:

- Memory address space
- I/O address space
- Interrupt level
- Arbitration level.

The assignment and uses of specific control areas in I/O space for each interface type are included as part of the architecture definition of each unit type.

Signalling Control areas use shared-memory and are used by the delivery level in conjunction with the signalling protocols. These operations are defined in "Signal" on page 2-5.

Physical Interface

An adapter can support either the Locate or Move mode architecture using hardware with the same set of control areas in I/O address space and with setup support to tailor the operation of the interrupt-valid bit and reset the protocol to match either mode.

At the physical level, the Move mode architecture defines the number and location of the control areas in I/O address space and the rules or protocols for their use in:

- Establishing access to the interrupt
- Causing the interrupt to an adapter
- Identifying the cause of the interrupt
- Resetting the interrupt
- Establishing addressability for the push or pull operation.

The base address of the I/O address space is defined during system configuration (Setup). The control areas within that I/O address space for a unit with a physical interface are defined in "Physical Interface Support" on page 1-15.

Note: Before using a unit with a physical interface, the enable-interrupt and the enable-bus-master bits in the Subsystem Control port (defined in "Subsystem Control Port" on page 1-17) must be set to 1.

Feature Adapter to System Unit Protocols

The following section defines the protocols used with the physical interface for the push, pull, and signal operations.

- Pushing and Pulling

  The protocol used to push and pull to or from shared memory is done differently depending upon whether it is being done from a system unit or adapter.

  - A system unit uses either Load and Store instructions or DMA operations to push and pull data, depending on the data being transferred.
  - Adapters perform the push and pull operation to or from shared memory using the bus-master function when the shared memory is not physically located on the adapter.

- Signalling (which causes a hardware interrupt)

  The protocol used to generate an interrupt to a system unit differs from that used to generate an interrupt to an adapter.

  - An interrupt of a system unit by an adapter or another system unit is caused by raising the interrupt level (IRQ line) defined at system configuration time (Setup).

    Because system units allow an interrupt level to be shared by several adapters, Signalling Control areas are used to identify the unit that generates the interrupt.

    The interrupt-valid bit in the Command Busy/Status port (defined in "Command Busy/Status Port" on page 1-18) also indicates which unit generated the interrupt.

  - An interrupt of an adapter is caused by writing to the Attention port. The reason code of hex D and the device number of hex 0 must be written to the Attention port as defined in "Attention Port" on page 1-16. Only reason code D and device number 0 are used by the physical protocol.

    When the Attention port is written to, the operation-pending bit in the Command Busy/Status port is set to 1. The receive-signal logic in the adapter is responsible for resetting this bit. Interrupt support in the adapter-control program invokes the receive-signal logic .

    The send-signal logic and receive-signal logic operate in support of the signalling protocols defined at the delivery level.

    - The send-signal logic (at the physical level) is used by the send-signal logic (at the delivery level) after it has pushed the signalling information into the appropriate Signalling Control area.

    - The receive-signal logic examines all Signalling Control areas when a hardware interrupt is received. This ensures that all signalling indications are captured even though a hardware interrupt could have been missed by some hardware-busy or masked condition.

      Using the information in the Signalling Control areas, the receive-signal logic posts the appropriate delivery-signal logic (at the delivery level) and clears the contents of the Signalling Control area to 0. For management, it posts

the unit-management logic and resets the management indicator in the appropriate Signalling Control area.

Note: In order to prevent the loss of interrupts when the operation-pending bit in the Command/Busy Status port is set to 1, the processing of any interrupt should include the processing of all Signalling Control areas. Then, following the reset of the operation-pending bit, a final recheck of all the Signalling Control areas should be performed.

Feature Adapter to Feature Adapter Protocols

Push, pull, and signal operations are also defined between two adapters, each having a physical interface.

The push and pull operations use the bus-master-DMA operations in both directions.

The signal operation (which causes a hardware interrupt) is done in the same manner as defined from adapter to system unit. The Signalling Control area identifies the cause and source of the interrupts.

Chapter 3. Delivery Level

The delivery level defines the services and protocols for managing the delivery of one or more control elements between a system unit and adapter or between adapters. The delivery level uses the services provided by the underlying physical level.

The primary function of the delivery level is to support the sending and receiving of control elements. Control elements are exchanged using a pair of delivery pipes. One delivery pipe is used for sending control elements, the other for receiving them.

The delivery level provides a set of services that can place control elements into a delivery pipe (enqueue), take control elements out of a delivery pipe (dequeue), and signal between units.

The delivery level defines the protocols used in the delivery of control elements. These protocols operate with the control areas located in shared memory, and are built on top of the physical-level support.

Structure

Figure 3-1 on page 3-2 shows the delivery-level structure for a configuration supporting control element delivery between a pair of units. The Local Enqueue and Local Dequeue Control areas are in memory that is local to each unit, and the Surrogate Enqueue and Surrogate Dequeue Control areas are in shared memory. The pipes, one in-bound and one out-bound, are also located in shared memory. There can be multiple pairs of delivery pipes in a user's system containing multiple units (system units and adapters).

Figure 3-1 on page 3-2 also shows that delivery support consists of enqueue logic, dequeue logic, and management logic and provides a set of services to these entities through the send and receive interfaces.

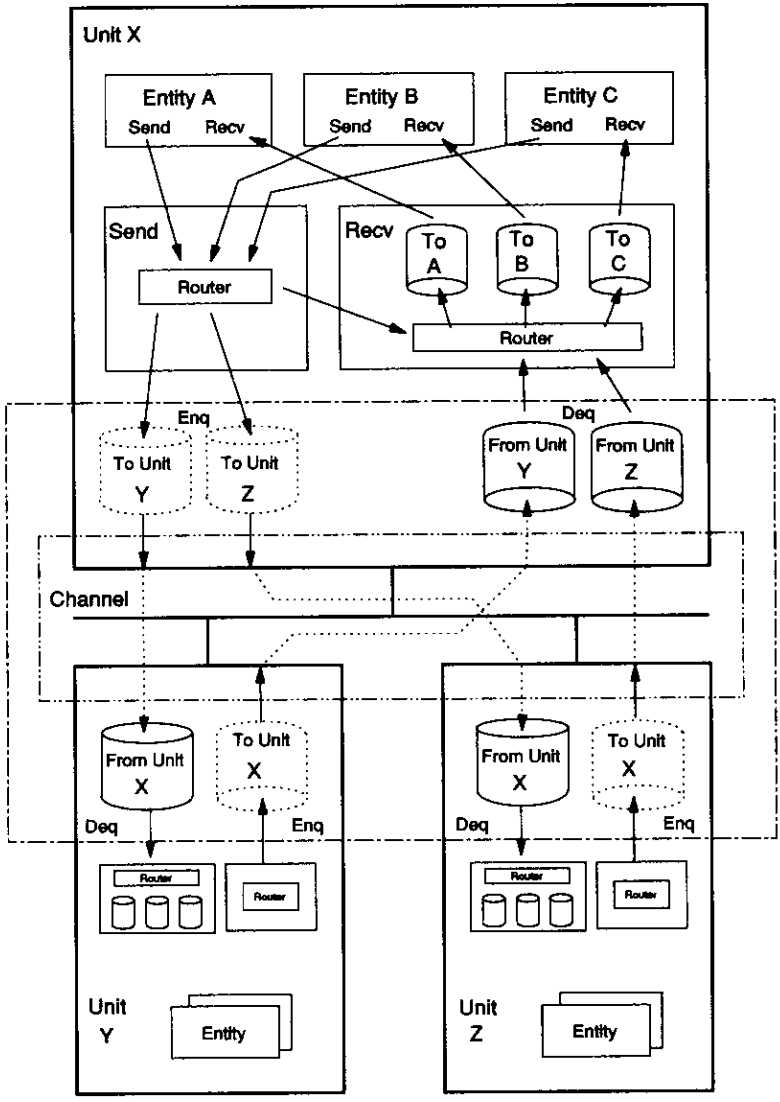

Figure 3-1. Overall Delivery Level Structure

Delivery Pipes

The following figure shows the relationship of the delivery pipe to the enqueue logic and dequeue logic (at either end of the delivery pipe) and the individual control areas associated with each pipe. For more information on the delivery pipe and the associated control areas, see "Delivery Pipe" on page 1-19.

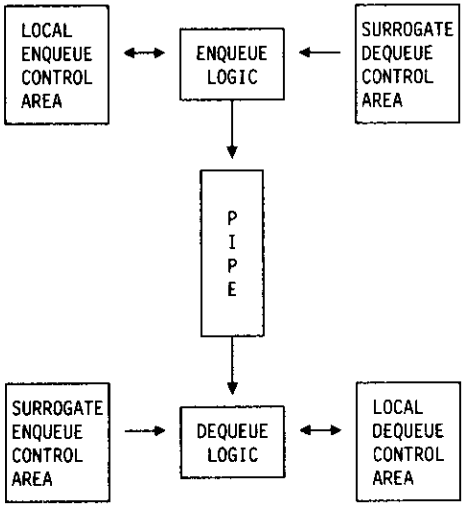

Figure 3-2. Delivery Pipe and Associated Control Areas

The enqueue logic places the control elements into the delivery pipe at the source; the dequeue logic removes the control elements from the delivery pipe at the destination. When a system contains several units, each pair of units has its own pair of delivery pipes (one in-bound and one out-bound). An example of a multiple unit structure is shown in the following figure.

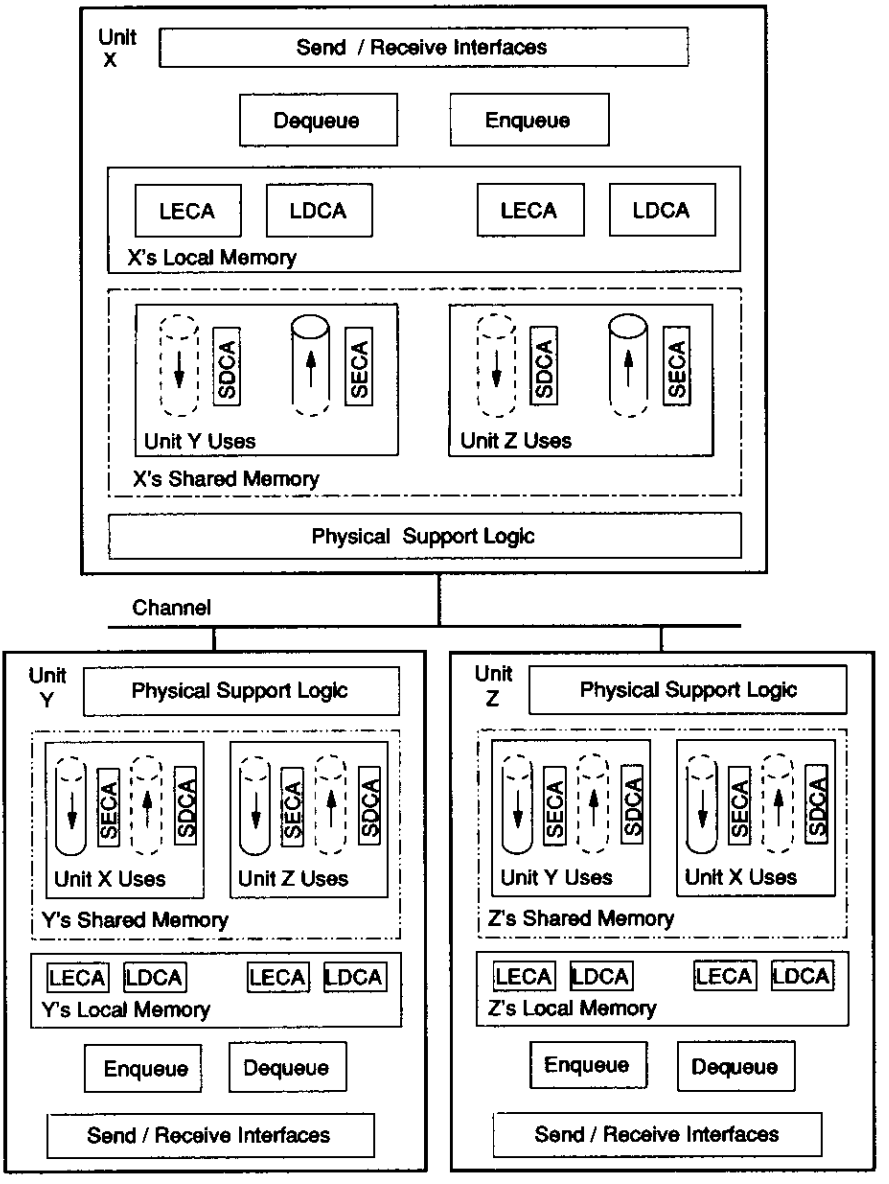

Figure 3-3. Multiple Unit Structure

In the example, the control areas and the delivery pipes are shown within dotted lines to indicate that they are located in shared memory.

Signalling

The enqueue and dequeue protocols operate using status information maintained in shared memory and do not require signalling operations. However, for certain implementations and protocols, there might be a need to signal the other unit when certain changes of the delivery pipe occur. The following are optional signalling conditions:

- When a delivery pipe goes from empty to not empty
- When a delivery pipe goes from not empty to empty
- When a delivery pipe goes from full to not full
- When a delivery pipe goes from not full to full
- When an control element is placed into a delivery pipe
- When an control element is removed from a delivery pipe.

The Move mode architecture also defines signalling protocols which support the management of the delivery-support logic.

Send and Receive Interfaces

Figure 3-1 on page 3-2 shows the send-interface logic between an entity sending a control element and the enqueue logic. The send-interface logic separates the various forms of operating-support interfaces and scheduling policies from the enqueue protocols used between units.

Figure 3-1 on page 3-2 shows the receive-interface logic between an entity receiving a control element and the dequeue logic. The receive-interface logic separates the various forms of operating-support interfaces and scheduling policies from the dequeue-logic protocols used between units.

Delivery Level Services

The services provided by the delivery level are used by entities at the processing level to exchange requests and replies in the form of control elements between correspondent entities. They include services to:

- Enqueue control elements
- Dequeue control elements.

Enqueue Service

The enqueue service is used to place one or more variable-length control elements into the delivery pipe. The enqueue logic uses the services provided by the underlying physical level to access the control areas associated with the destination unit and to place control elements into the delivery pipe in shared memory.

The specifics for calling this service are an internal function and are not defined by the architecture. However, the following illustrates the parameters required when calling this service:

```
ENQUEUE (address, count, urgency, return code)
```

The ***address*** parameter indicates the location in the address space of the caller of the control element to be placed into the delivery pipe.

The ***count*** parameter indicates the number of control elements to be enqueued. If more than one control element is specified, all the control elements must be contiguous to one another in the memory space identified by the address parameter.

The ***urgency*** parameter indicates the relative priority of the request to enqueue a control element: normal or expedited.

The ***return code*** parameter is returned to the caller and indicates the success or failure of the enqueue operation.

The Destination field in the control element determines the destination unit and the delivery pipe that deliver the control element.

The Source field in the control element identifies the source unit and delivery pipe used when returning a reply.

Dequeue Service

The dequeue service is used to initiate the removal of control elements from the delivery pipe. Control elements are always removed from the head of the delivery pipe. The dequeue logic uses the services provided by the underlying physical level to access the control areas associated with the delivery pipe, as well as the delivery pipe.

Control elements are passed to the receive-interface logic using a call-back mechanism. Using this approach, the receive-interface logic calls the dequeue logic, which in turn calls back the receive-interface logic with the address and length of the control element at the head of the delivery pipe. The receive-interface logic uses the contents of the control element to determine which entity receives the control element and how to handle the delivery of the control element to the destination entity.

This process repeats itself until the delivery pipe is empty or until the receive-interface logic indicates it cannot accept any more control elements.

Local policy determines the means of delivery. The control element can be passed directly to the entity, or it can be queued for processing later. If the entity is not active, does not exist, or does not have an active read pending, the control element can be returned to the sender.

The specifics for calling this service are an internal function and are not defined by the architecture. However, the following conceptual service illustrates the parameters required when calling the dequeue service:

```
DEQUEUE (unit ID, return code)
```

The ***unit ID*** parameter identifies which delivery pipe is used.

The ***return code*** parameter is returned to the caller and indicates the success or failure of the service call.

Delivery Level Protocols

The delivery-level protocols define the operations of the delivery-level support as they appear across the channel. This includes defining the push and pull operations used with the physical-level services, and the signalling operations used with the signalling services.

The operation of the delivery-level protocols is defined for a given pipe. At the delivery level, all pipes operate the same. An overview of the push and pull operations between the enqueue logic and its control areas and the dequeue logic and its control areas is shown in Figure 3-4 on page 3-9. The specific definitions for these control areas can be found in "Delivery Pipe" on page 1-19. The locations in shared memory for the control areas shown are defined at initialization by the configuration records. The layout of the configuration record is shown in Figure 5-1 on page 5-5.

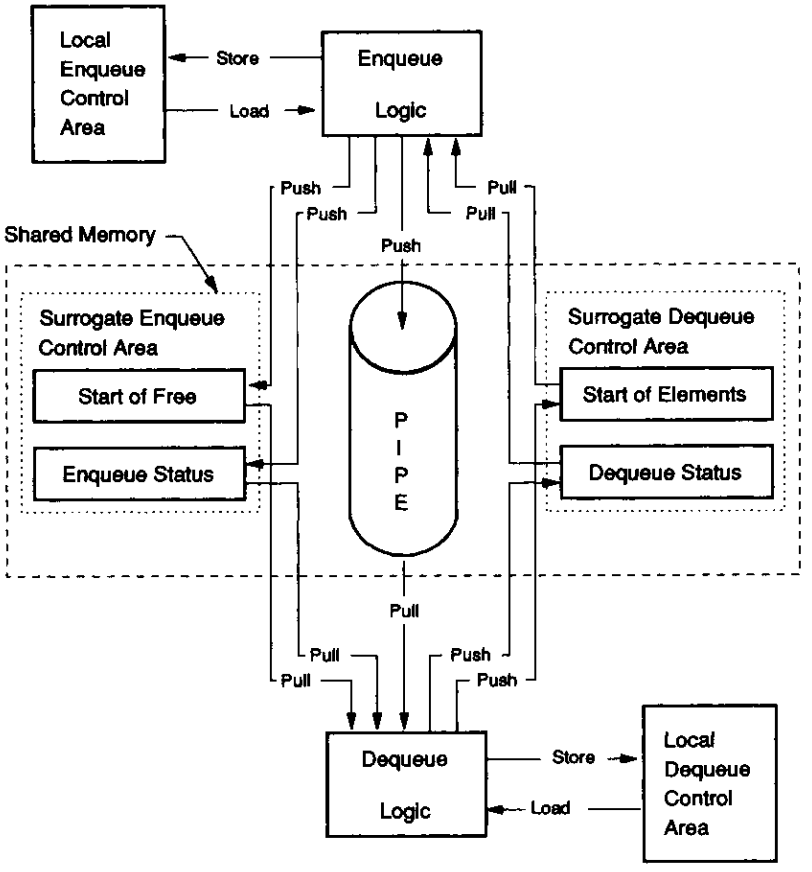

Figure 3-4. Delivery Pipe and Associated Control Areas

The overall operation of the signalling portion of the enqueue and dequeue logic is shown in Figure 3-5 on page 3-10. The specific definition of the Signalling Control area can be found in "Signalling Control Area" on page 1-29.

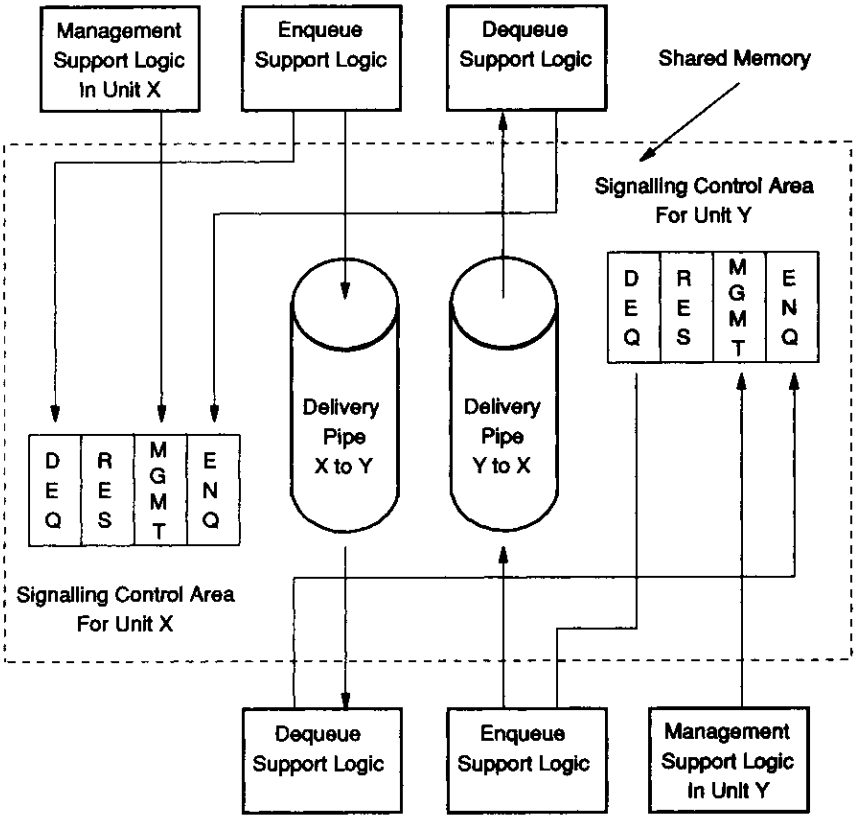

Figure 3-5. Signalling Control Area Use

Each delivery pipe has a set of pointers in the Local Enqueue Control area and in the Local Dequeue Control area. The pointers in the Local Enqueue Control area indicate the start-of-free space and the end-of-free space. The pointers in the Local Dequeue Control area indicate the start-of-elements and the end-of-elements. Both control areas have pointers that indicate the limits of the delivery pipe:

- TOP - the threshold point for normal enqueue operations.
- BASE - the beginning of the usable space in a delivery pipe.
- END - the end of the usable space.

Examples of the delivery pipe in different states are shown in the following figure. An example of a control element that is being wrapped is included. This occurs when a control element does not fit into the space remaining between the current end-of-free space (EF) and the top-of-pipe (TOP).

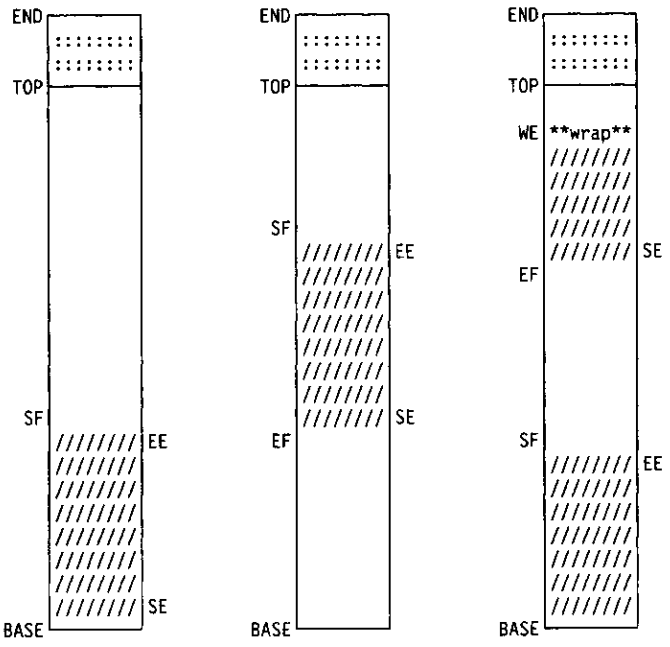

Figure 3-6. Various Examples of Pipe Use

The following sections define the specific protocols for the enqueue logic and dequeue logic. The logic is defined in pseudo code, which operates on the information in the control areas and calls the services of the physical level.

Enqueue Protocol

The protocol defines the operation of the enqueue logic that provides control elements with the enqueue service.

The following discusses how the enqueue logic uses the control areas in shared memory to manage the operation of the delivery

pipe. The individual bits and fields are described in "Delivery Pipe" on page 1-19 and "Signalling Control Area" on page 1-29.

- The size of a control element is specified in bytes and can be found in the Length field of the control element. Several control elements can be enqueued with a single invocation of the enqueue service. However, all of the control elements involved must be contiguous in memory (see "Enqueue Service" on page 3-6).
- The location of the top-of-pipe (TOP) is determined by the implementation but must be sufficient to hold at least one wrap control element.
- The enqueueing of a control element is a local operation when the delivery pipe is physically located in the source unit. It is a distributed operation and involves a push of the control element across the channel when the delivery pipe is physically located in the destination unit.
- The type of operation (local or distributed) required to access the Surrogate Start of Free and Surrogate Enqueue Status fields depends on where they are located in shared memory.
- For the enqueue logic, the receive-signal logic is used to synchronize the state of the empty bit in the Surrogate Dequeue Control area with the state of the empty bit in the Local Enqueue Control area.
- To allow the tailoring of signalling conditions, several signalling options can be specified during configuration and initialization. These options are used by the state-change function of the enqueue logic to indicate the conditions under which a signal is sent by the enqueue logic to the dequeue logic at the other end of the delivery pipe. The following describes the signalling-option bits and the actions taken when they are set to 1.
  - If the not-full to full option is selected, a signal is sent when the pipe goes from a not-full state to a full state.
  - If the empty to not-empty option is selected, a signal is sent when the pipe goes from an empty state to a not-empty state.
  - If the signal-on-enqueue option is selected, a signal is sent each time a control element is placed into the pipe.

    Note: Care should be exercised when configuring the signal on-enqueue option. Use of this option with high-speed

adapters could result in an overload of the destination unit's interrupt handler.

- Several bits in the Enqueue Status field of the Local Enqueue Control area are used internally to control the operation of the enqueue logic.
  - The queued bit, when set to 1, indicates a control element has been successfully placed into the delivery pipe.
  - The empty bit, when set to 1, indicates the empty bit in the Surrogate Dequeue Status field is set to 1.
- Several other bits in the Enqueue Status field of the Local Enqueue Control area hold information used to update the Status field in the Surrogate Enqueue Control area.
  - The full-status bit, when set to 1, indicates the delivery pipe is full from the perspective of the enqueue logic.
  - The wrap bit is used to indicate that a wrap control element has been placed into the delivery pipe by the enqueue logic. The enqueueing of a wrap control element causes a toggling of this bit (1 to 0 or 0 to 1).

The pseudo code in the following pages is an example of the enqueue protocols. These protocols are:

- Enqueue initialization
- Enqueue receive signal
- Enqueue function
- Enqueue state-change function.

Enqueue Initialization

The following occurs when the delivery service in a unit is initialized.

```
{Setup the fields in the local enqueue control area containing
 the starting and ending offset values for start_of_free and
 end_of_free.}

SET start_of_free = BASE
SET end_of_free = top_of_pipe

{Setup various flag bits in the enqueue status field of the
 local enqueue control area to their correct initial value.}

SET enqueue_wrap = 0
SET full = 0
SET queued = 0
SET empty = 1

{Setup the field in the surrogate dequeue control area containing
 the starting offset value for start_of_elements.}

SET surrogate start_of_elements = BASE

{Setup the various flag bits in the dequeue status field of the
 surrogate dequeue control area to their correct initial value.}

SET surrogate dequeue_wrap = 0
SET surrogate empty = 1
```

Enqueue Receive Signal

The following occurs when a signal is received from the dequeue logic at the other of the delivery pipe. This is done prior to invoking the enqueue logic.

```
IF surrogate empty = 1 THEN
   SET empty = 1
ENDIF
```

Enqueue Function

The following pseudo code represents the main-line code of the enqueue-support logic. It is called by the send-interface or receive-interface logic to place control elements into a delivery pipe. See "Enqueue Service" on page 3-6 for a description of the calling parameters and the return codes.

```
PROCEDURE Enqueue (control element address, number, urgency, return code)

   {Call the queue routine to place a control element in the pipe
    for as long as there are control elements to enqueue or until
    an exception condition is encountered. }

   DO
      CALL queue_element (control element address, return code)
      SET number = number - 1
   WHILE return = successful and number ≠ 0

   RETURN return code to caller
ENDPROC
```

The following pseudo code is called by the main-line code of the enqueue-support logic. It places control elements into the pipe one at a time, updates the appropriate Local and Surrogate Control areas, and signals the dequeue logic at the other end of the pipe, when necessary.

```
PROCEDURE queue_element(control element address, return code)

   {Update the end of free space (enqueue tail) by tracking
    where the start of elements (dequeue head) is currently
    located.}

   IF (end_of_free > surrogate start_of_element) OR
      (end_of_free = surrogate start_of_element AND
       enqueue_wrap = surrogate dequeue_wrap)
      THEN
         SET end_of_free = top_of_pipe
      ELSE
         SET end_of_free = surrogate start_of_elements
   ENDIF

   {Check to see if an element can be placed in the
    pipe.  If so, place it in the pipe, update the
    enqueue state information, and return indication
    that the enqueue was successful.}

   IF length of element ≤ end_of_free - start_of_free
      THEN

         {Element can be placed into the pipe.}

         move element onto pipe at start_of_free
         increment start_of_free by length of element
         SET queued = 1
         CALL state_change
         RETURN successful indication to caller
      ELSE

         {Element won't fit in the pipe.  Check if the pipe
          needs to wrap to the top_of_pipe.  If it does,
          build wrap control element, place it in the pipe,
          update the enqueue state information, and try
          again to place the element in the pipe.  If it
          does not, update enqueue state information and
          return pipe full indication to caller.}

         IF end_of_free = top_of_pipe
            THEN

               {A wrap condition exists.}

               push wrap element onto pipe at start_of_free
               SET end_of_free = surrogate start_of_elements
               SET start_of_free = base

               {Toggle the state of the wrap bit.}

               IF enqueue_wrap = 0
                  THEN
                     SET enqueue_wrap = 1
                  ELSE
                     SET enqueue_wrap = 0
               ENDIF
```

```
                {Now check to see if control element will fit
                 in the pipe after we have wrapped to BASE of
                 the pipe.  If it does, move control element
                 into the pipe and update the enqueue state
                 information.  If not, update state information
                 and return full condition to caller. }

                IF element length ≤ end_of_free - start_of_free
                   THEN

                      {Element will fit in the pipe.}

                      push element onto pipe at start_of_free
                      increment start_of_free by element length
                      SET queued = 1
                      CALL State_Change
                      RETURN successful indication to caller
                   ELSE

                      {Element still won't fit in the pipe.

                      SET full = 1
                      CALL State_Change
                      RETURN pipe full indication to caller
                ENDIF

             ELSE

                {Pipe has not wrapped, but it is full.  Update
                 the enqueue state information and indicate
                 control element not queued.}

                SET full = 1
                CALL State_Change
                RETURN pipe full indication to caller
          ENDIF
    ENDIF
ENDPROC
```

Enqueue State Change Function

The following pseudo code represents the state-change code of the enqueue-support logic. It is called by the enqueue function to update the status information associated with the Surrogate Enqueue Control area, and to determine if a signal must be sent to the unit at the other end of the delivery pipe.

```
PROCEDURE state_change()

   {Update the surrogate enqueue control area.}

   Move enqueue_status to surrogate enqueue_status
   Move start_of_free to surrogate start_of_free

   {Determine if a change in the state of the pipe has
    occurred which requires the sending of a physical
    signal to the dequeue logic in the unit at the other
    end of the pipe.}

   IF (queued = 1 AND signal on-enqueue = 1) OR
      (queued = 1 AND surrogate empty = 1 AND
      empty to non-empty = 1) OR
      (full = 1 AND not-full to full = 1)
      THEN

         {Physically send a signal to the dequeue logic in
          the specified unit indicating a change in state.}

         CALL signal(unit, dequeue logic, state_change)
   ENDIF
   SET empty = 0
   SET full = 0
   SET queued = 0
ENDPROC
```

Dequeue Protocol

The dequeue protocol defines the operation of the dequeue logic that provides the control element dequeue service.

The following discusses how the dequeue logic uses the control areas in shared memory to manage the operation of the delivery pipe. The individual bits and fields are described in "Delivery Pipe" on page 1-19 and "Signalling Control Area" on page 1-29.

- The dequeueing of a control element is a local operation if the delivery pipe is physically located in the destination unit. It is a distributed operation and involves a pull across the channel when the delivery pipe is physically located in the source unit.
- The type of operation (local or distributed) required to access the Surrogate Control areas depends upon where they are physically located in shared memory.
- The dequeue logic removes and discards the wrap control element when dequeueing so that the space between the wrap control element and the top-of-pipe (if any) can be made available to the enqueue logic.
- For the dequeue logic, the receive-signal logic is used to synchronize the state of the full bit in the Surrogate Enqueue Control area with the state of the full bit in the Local Dequeue Control area prior to invoking the dequeue logic.
- To allow the tailoring of signalling conditions, several signalling options can be specified during configuration and initialization. These options are used by the state-change function of the dequeue logic to indicate the conditions under which a signal is sent by the dequeue logic to the enqueue logic at the other end of the delivery pipe. The following describes the signalling-option bits and the actions taken when they are set to 1.
  - If the not-empty to empty option is selected, a signal is sent when the pipe goes from a not-empty state to an empty state.
  - If the full to not-full option is selected, a signal is sent when the pipe goes from a full state to a not-full state.
  - If the signal-on-dequeue option is selected, a signal is sent each time a control element is removed from the pipe.

Note: Care should be exercised when configuring the signal on-dequeue option. Use of this option with high speed adapters could overload the source unit's interrupt handler.

- Several bits in the Dequeue Status field of the Local Dequeue Control area are used internally to control the operation of the dequeue logic.
  - The dequeued bit, when set to 1, indicates a control element has been successfully removed from the delivery pipe.
  - The pre-empt bit, when set to 1, indicates the current dequeue operation is terminated even if the delivery pipe is not empty. The pre-empt bit is set based on the return code from the receive-interface logic. A value of 0 indicates that another control element can be passed to this logic. A value of 1 indicates that no more control elements are passed at this time.
- Several other bits in the Dequeue Status field of the Local Enqueue Control area hold information used to update the Status field in the Surrogate Dequeue Control area.
  - The empty bit, when set to 1, indicates the delivery pipe is empty from the perspective of the dequeue logic.
  - The wrap bit is used to indicate that a wrap control element has been removed from the delivery pipe by the dequeue logic. The dequeueing of a wrap control element causes a toggling of this bit (1 to 0 or 0 to 1).

The pseudo code in the following pages is an example of the dequeue protocols. These protocols are:

- Dequeue initialization
- Dequeue receive signal
- Dequeue function
- Dequeue state-change function.

Dequeue Initialization

The following occurs when the dequeue portion of the delivery service in a unit is initialized.

```
   {Setup the fields in the local dequeue control area containing
    the starting and ending offset values for start_of_elements and
    end_of_elements.}

   SET start_of_elements = BASE
   SET end_of_elements = BASE

   {Setup the various flag bits in the dequeue status field of the
    local dequeue control area to their correct initial value.}

   SET dequeue_wrap = 0
   SET pre-empt = 0
   SET dequeued = 0
   SET full = 0
   SET empty = 1

   {Setup the field in the surrogate enqueue control area containing
    the starting offset value for start_of_free.}

   SET surrogate start_of_free = BASE

   {Setup the various flag bits in the enqueue status field of the
    surrogate enqueue control area to their correct initial value.}

   SET surrogate enqueue_wrap = 0
   SET surrogate full = 0

ENDPROC
```

Dequeue Receive Signal

The following occurs when a signal is received from the enqueue logic at the other of the delivery pipe. This is done prior to invoking the dequeue logic.

```
IF surrogate full = 1 THEN
   SET full = 1
ENDIF
```

Dequeue Function

The following pseudo code represents the main-line code of the dequeue-support logic. It is called by the receive-interface logic to initiate removing control elements from a delivery pipe. See "Dequeue Service" on page 3-7 for a description of the calling parameters and the return codes.

```
PROCEDURE dequeue (unit, return code)

   IF start_of_elements = surrogate start_of_free AND
      dequeue_wrap = surrogate enqueue_wrap
      THEN
         CALL state_change
         RETURN pipe empty to caller
      ELSE

         {Remove all the elements from the pipe and pass them one at
          a time to the input router of the receive interface logic.}

         DO
            SET end_of_elements = surrogate start_of_free

            {Check the element at the head of the pipe.  If it is a wrap
             element, discard it, reset the start of elements pointer to
             the base of the pipe, and toggle the setting of the dequeue wrap
             state.  If not, call the input router function of the receive
             interface logic with the address of the element.}

            IF element at start_of_elements is a wrap element
               THEN
                  dequeue and discard the wrap element
                  SET start_of_elements = base of pipe
                  SET dequeued = 1

                  {Toggle the dequeue wrap state.}

                  IF dequeue_wrap = 0
                     THEN
                        SET dequeue wrap = 1
                     ELSE
                        SET dequeue wrap = 0
                  ENDIF
               ELSE
                  CALL receive_input_router (start_of_elements)
                  IF return_code = 1 THEN
                     SET pre-empt = 1
                  SET start_of_elements = start_of_elements + element length
                  SET dequeued flag = 1
                  CALL state_change
            ENDIF

         WHILE (pre-empt = 0) AND
                (start_of_elements ≠ end_of_elements OR
                 dequeue_wrap ≠ surrogate_wrap)

         {Determine if we stopped because pipe was empty.  If so,
          set empty and call state_change.}

         IF pre_empt ≠ 1 THEN
            SET empty = 1
            SET pre-empt = 0
            CALL state_change
         ENDIF
   ENDIF
ENDPROC
```

Dequeue State Change Function

The following pseudo code represents the state-change code of the dequeue-support logic. It is called by the dequeue function to update the status information associated with the Surrogate Dequeue Control area, and to determine if a signal must be sent to the unit at the other end of the delivery pipe.

```
PROCEDURE state_change ();

   {Update the surrogate dequeue control area.}

   Push dequeue_status to surrogate dequeue_status
   Push start_of_elements to surrogate start_of_elements

   {Determine if a change in the state of the pipe has
    occurred which requires the sending of a physical
    signal to the enqueue logic in the unit at the other
    end of the pipe.}


   IF (dequeued = 1 AND signal on dequeue = 1) OR
      (surrogate full = 1 AND full to not-full = 1) OR
      (empty = 1 AND not-empty to empty = 1)
      THEN

         {Physically send a signal to the enqueue logic in
          the specified unit indicating a change in state.}

         CALL signal(unit, enqueue logic, state_change)
   ENDIF

   SET empty = 0
   SET full = 0
   SET dequeued = 0

ENDPROC
```

Notes:

Chapter 4. Processing Level

At the processing level, requests and the responses to those requests are exchanged between entities using the services provided by the underlying delivery level. These entities represent the clients and the servers in the various system units and adapters attached to the channel.

An entity sends a request to another entity or receives a reply to a previous request using control elements. The information contained in these control elements, their number, order of exchange, and meaning, constitute a protocol between the two entities. This protocol is used by the two entities to coordinate and control their activities. The architecture does not define these protocols. However, the architecture does define the structure and format of the control elements and the content and use of a common header that is required in all control elements.

The delivery service uses information in this common header to deliver control elements without knowing or understanding what is contained in the body of the control element.

The cooperating entities use information in the common header to identify the source, indicate the type and urgency of the information contained in the control element, and to correlate replies with previous requests.

Structure

The processing level is structured into two parts: an interface part and a processing part. The interface part provides a means for tailoring the interface between the entities at this level, the local operating environment, and the underlying enqueue and dequeue services provided by the delivery level. The processing part consists of the entities that use the underlying delivery service to communicate with one another in the performance of a common task.

The send-interface and receive-interface logics provide a means of tailoring the local interface in each unit (system unit or adapter) to

the processing requirements of the entities, their operating environment, and the underlying delivery service.

This allows the receive-interface logic to place control elements in a local queue until the destination entity is ready to process them, instead of requiring the destination entity to wait for the arrival of a control element.

The operating environment at one end of the delivery pipe might not support the ability to block or suspend the execution of an entity pending the receipt of a control element, while the operating environment at the other end of the delivery pipe might be able to do so.

Each of these situations has an impact on the local behavior of the communicating entities, but not on their ability to send or receive control elements to each other.

The content of control elements is totally dependent on the entity pair. However, the architecture defines a set of common-use control elements and protocols at the processing level when either command or data chaining is used. The architecture also makes provisions for defining additional control elements used between entities whose requirements cannot be met by the control elements in the common-use set.

Processing Level Operation

The delivery-support logic operates in conjunction with the various entities and the underlying operating environment. The following shows an example of this interaction.

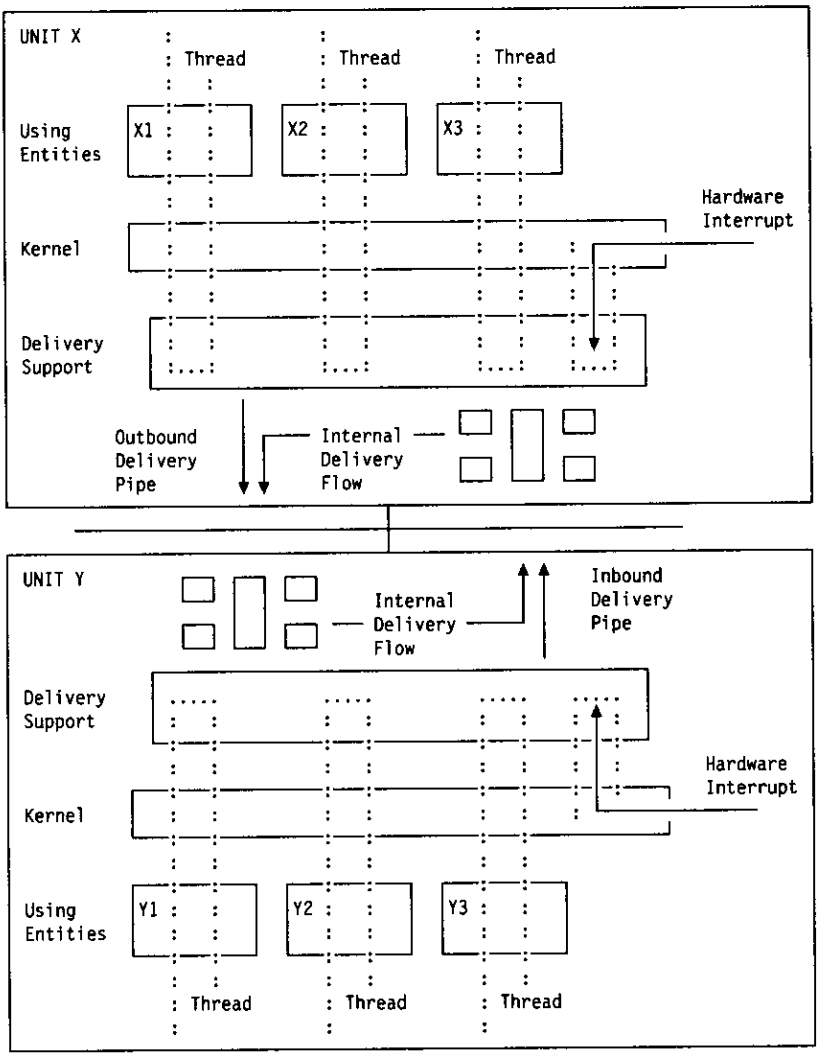

Figure 4-1. Device Driver or Handler Model

Each entity using the delivery support is operated on a unit of dispatching (called a thread). The delivery support is functionally similar to that of a device driver in operating-systems environments (Figure 4-2 on page 4-5).

- It has a strategy portion which runs under the same thread as the entity.
- It can allow only one thread at a time to use the delivery-support logic. This feature can be used to serialize the enqueueing of control elements into the shared-delivery pipe.
- It allows the delivery-support logic to cause a wait condition (block) on a specific thread. This can be used to handle pipe-full or receive-pending conditions.
- It has a local-data area for the delivery-support logic. This can be used for pipes as well as for delivery state and configuration.
- It has an interrupt handler which is called when a hardware interrupt occurs. This can be used to support the signalling protocol and initiate the dequeue logic.

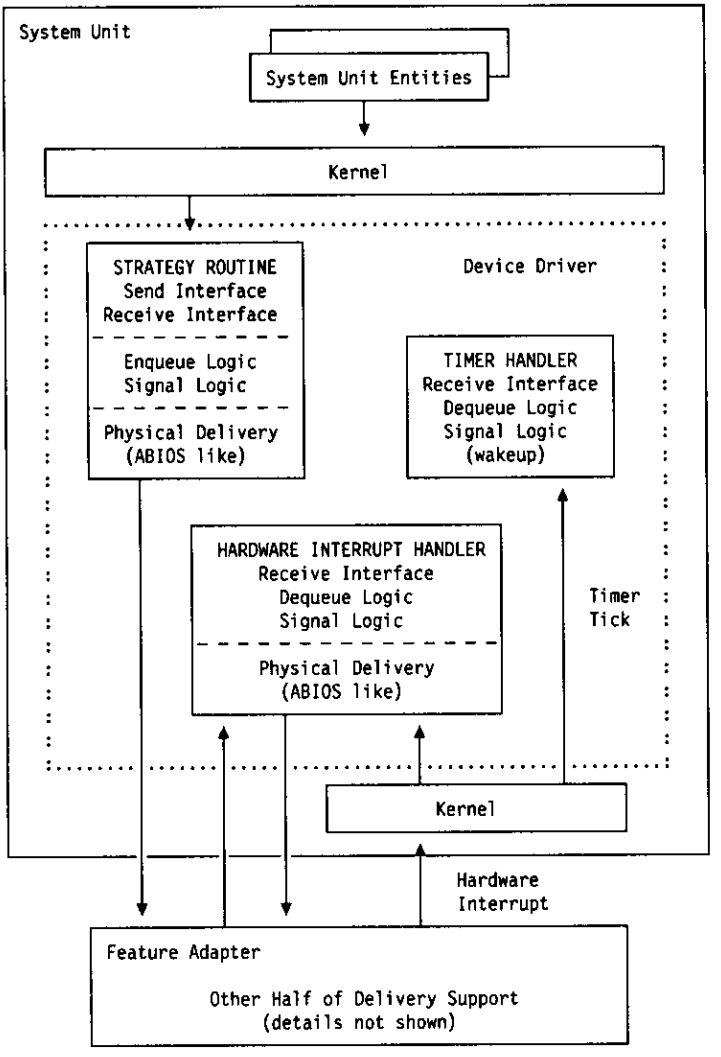

Figure 4-2. Device Driver or Handler Model

A hardware interrupt causes the scheduling of the signalling portion of the delivery support for that unit. The physical protocol that causes the interrupt differs for system units and adapters.

Figure 4-2 also shows that control elements can be removed from the pipe going in one direction between units and placed in the pipe going in the other direction between units (internal delivery flow).

This routing of control elements can be considered an internal operation. This capability is required to support the handling of certain error conditions. The figure does not show the physical-delivery support used to pass control elements, data, and signals between the two units.

Send

The enqueue operation is performed in the device driver or handler routine. The control elements move on the entity's thread when the Send command is executed. The details of the enqueue logic are defined in "Control Element Delivery" on page 1-11.

The architecture does not define policy. However, the send-interface logic provides a means of implementing local policy for each unit. The send-interface logic determines:

- Whether a single control element or multiple control elements are used.
- Whether control is returned directly to the calling entity when the pipe is full, or the thread is blocked until the full condition clears.

Receive

On the receive side, a receive pending exists for any Send command between an entity pair. The Send command can involve requests, replies to requests, or event type control elements. The specific entity-to-entity protocol determines how the Send and Receive commands are correlated. This correlation can be different for each entity pair. It is not restricted by the architecture.

Any queueing for the server is performed at the processing level. The delivery pipe is used only to mask the characteristics of the physical interface when multiplexing the delivery of control elements to multiple entities in the same unit. The model used for control element delivery is that of a free-flowing pipe running between the source and destination entities.

The receiving of control elements is handled the same as a read-pending operation. The space for the control element must be pinned and known by the receive-interface support so the control element can be moved from the delivery pipe into shared memory without performing a context switch. Placing the control element into

shared memory satisfies the receive pending at the receiving entity. If the thread with the receive pending was blocked, it becomes unblocked.

If the receiving entity does not have a receive pending for the control element sent, the receive-interface logic removes the control element from the delivery pipe, and either discards it or returns the common header (returning the header is device dependent).

When the control element is returned:

- The Source and Destination fields are reversed.
- The control element is set to an error type control element.
- The Status Information field is updated.
- The remainder of the request control element is discarded.
- The Length field is updated.

To prevent flooding of the delivery pipe in the opposite direction, and to prevent continuous wrapping of reply and error type control elements, additional request control elements and any other type of control elements received while a Receive command is not pending might be discarded. This prevents out-of-step protocols associated with one entity-to-entity flow from blocking the flow to other entities.

The space management for the delivery pipes can provide space within the pipe to accommodate this internal flow although the delivery pipe could be considered full for normal flow.

The architecture does not define what makes up a receive pending. It can be handled in a number of different ways because it depends on each unit's receive-interface-support logic. For example, some implementations choose to have the receive-interface logic maintain a set of queues for control elements. There might be a single queue for all entities at the destination, or there might be a separate queue for each entity. These queues can be arranged by control element type (request, reply, error, event), entity, or any other manner.

Interrupt

The use of the device driver or handler model for the control element delivery allows the hardware interrupts at the physical level to be separated from the control element delivery provided to entity-to-entity pairs.

Entity-to-entity protocols can operate using request, reply, error, and event control elements which are passed through a pair of delivery pipes. This separates the operation of the entity-to-entity protocols from the hardware interrupts.

The delivery-level state information for the enqueue and dequeue protocols is defined so that the pipe can operate without interrupts if the support observes the changes in state and acts accordingly.

Hardware interrupts are used to assist certain implementations in keeping the control elements in pipes flowing between units. Separating requests from replies allows interrupt overhead to be reduced because several control elements are dequeued when a single hardware interrupt is handled.

The delivery-level protocols related to interrupts are called signals. The architecture defines the signalling protocols if they are required by the implementation. Signalling can be done at user specified intervals to maintain the flow in the pipe and when certain state changes occur within the delivery pipe. The specific state changes are defined as part of the architected enqueue and dequeue protocols.

Processing Level Services

The Move mode architecture does not define the services used at the processing level. It only provides a means for tailoring the interfaces between the entities at this level, the local operating environment, and the underlying delivery level.

The functions provided by the send and receive interfaces depend on the implementation. Individual implementations can consider sending multiple control elements with a single call, and can decide to return on a pipe-full condition or wait for the condition to clear.

The receive-interface logic can provide the queueing of control elements for entities that do not have a read pending. Each entity can have its own individual queue or multiple queues corresponding to the type of control element (request, reply, error, event).

Processing Level Protocols

In general, the processing-level protocols are not defined by the architecture. However, the architecture does define common uses for the control elements used in the entity-to-entity flow. These include:

- Command chaining
- Data chaining
- Notification and wait
- Initialization (peer level).

Command Chaining

The server can process the control elements synchronously or asynchronously, depending on whether commands are being chained or not. Normally, a server can process requests in any order (asynchronously).

Command chaining is used to force the server to process control elements synchronously. When the command-chaining bit is set, the processing of the next control element in the chain cannot begin until the server completes the processing of the current control element and returns the appropriate reply to the client. The chaining process continues until all control elements in the chain are processed or an error is detected. When an error is detected, an error type control element is returned and the remaining control elements in the chain are discarded.

Data Chaining

Data chaining is a method of managing complex data transfers. For example, a single logical block of data can be located in several noncontiguous areas in shared memory and might need to be transferred to a single destination. Or, a single block of data might need to be transferred from a single source and placed in several noncontiguous areas of shared memory.

The architecture supports two methods of data chaining: direct and indirect.

Direct Data Chaining

In direct data chaining, each noncontiguous data area is described by an entry in a chaining list that is included in the request control element. The entry contains a byte count and an address pointer to a data area. After each data area described by an entry is transferred, the next entry in the list is processed. This action continues until all the entries in the list have been processed or until no more data exits. Direct data chaining is invoked by using the read list and write list function codes (see "Read List" on page 1-43 and "Write List" on page 1-49).

Indirect Data Chaining

Indirect data chaining is similar to direct data chaining, except for the location of the chaining list. The list, however, is not part of the request control element. Instead, the list is kept separate and is referenced by means of an indirect list pointer in the Value field of the request control element. Indirect data chaining is invoked by setting the indirect bit in a read or write list request control element. The following figure illustrates how the request control element and indirect list appear for a read list request.

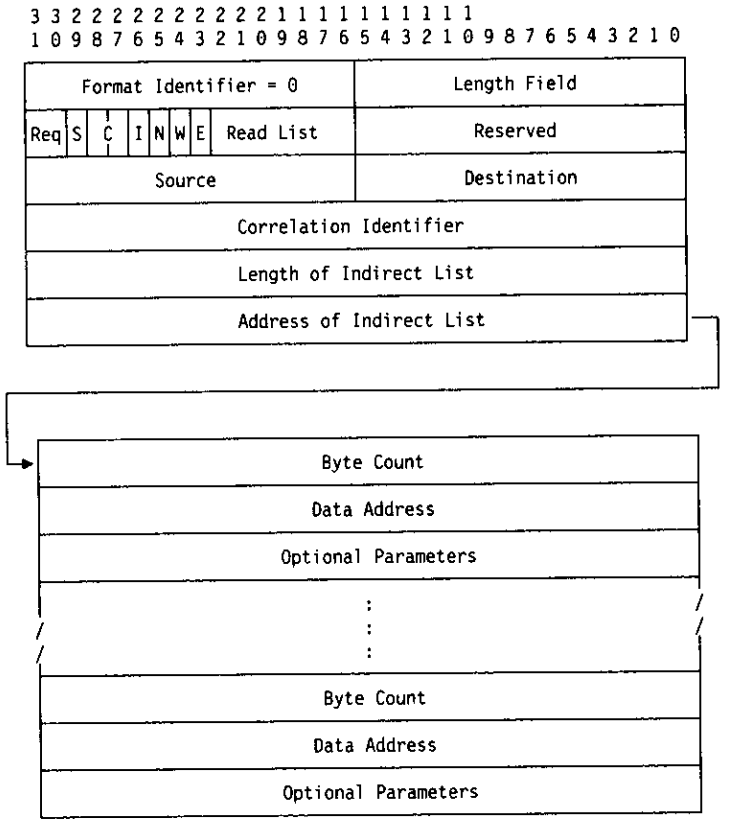

Figure 4-3. Indirect Data Chaining

Notification and Wait

A client can request notification when a server is about to process a specific control element by setting the notification bit (N) in that control element. The server then returns an event control element with the Source, Destination, and Correlation Identifier fields for the appropriate control element to the client.

A client can suspend the processing of a chain of control elements at a particular point by setting the wait bit (W) in the appropriate control element prior to submitting it to the delivery service. This causes the server to:

- Suspend further processing of request control elements

- Return an event notification control element that contains the Correlation Identifier field of the control element with the suspend bit in its Value field
- Wait for a resume event from the originating client before continuing.

The original request control element that contains the wait bit is not processed until the resume event is received from the client.

The following table shows the actions taken for different combinations of the wait and notification bits.

Notify	Wait	Action
0	0	Continue processing request control elements.
0	1	Suspend processing of the current control element and all subsequent request control elements from this source until a resume event control element is received from this source.
1	0	Return a notification to this source and process the request control element and all subsequent control elements from this source.
1	1	Return a notification to this source and suspend processing of the current control element and all subsequent control elements from this source until a resume event control element is received from this source.

Figure 4-4. Notify and Wait Actions

Chapter 5. Design Considerations

This section discusses the following areas, which are part of the Move mode architecture definition.

- Configuration
- Initialization
- Exception handling
- System management.

Configuration

Proper configuration of the system and the subsequent initialization of the individual units (system units and adapters), as well as the delivery service itself, are prerequisites for using the services. As such, it is considered to be a part of the Move mode architecture.

The process of initializing the delivery services requires that the following configuration information be defined and available at the time the system is loaded:

- Unit level
- System level
- Peer level.

Unit Level

Information from the unit-level configuration is specific to a single unit (system unit or adapter). The type of information provided is similar to that in an adapter description file (adapter description file is defined by the Setup Architecture). This includes, but is not limited to, the following information:

- Adapter ID
- Adapter base I/O address
- Adapter base I/O size
- Adapter base shared-memory address
- Adapter base shared-memory size
- Interrupt level
- DMA arbitration level.

System Level

Information from the system-level configuration allows a pair of cooperating units (system or adapter) to communicate with each other. This information is used to set up and coordinate the delivery service. This information includes, but is not limited to the following:

- Unit ID

  The delivery support uses a Unit ID to associate a unit with the set of control areas in memory and I/O address space.

- Peer Unit ID

  At the system level, each unit requires an ID and IDs of the other units it sends control elements to or receives control elements from.

- Unit Type

  The delivery service supports the following operations:

  - System unit to adapter
  - Adapter to adapter
  - System unit to system unit.

  Both units must identify the configuration information for the unit type (system unit or adapter).

- Resource Definitions

  The delivery pipes require:

  - Configuration information describing the size and location of the delivery pipe
  - Operational information defining the use of the pipe
  - Information indicating the current state of the pipe.

  The following is a summary of the specific information required to allocate and use these resources:

  - Placement

    The pipe can be located anywhere in shared memory.

  - Size and location

    Both units need to know the size and location of the delivery pipes and the associated control areas.

- Signalling Control area locations

  The method of signalling between units is determined by the unit type and unit pairing (system unit to adapter, adapter to system unit, or adapter to adapter.). Both units must have the location of the Signalling Control area before signalling the other unit.

- Signalling conditions

  The delivery level supports a number of signalling options. Both units must be configured to support the options used. The following are some optional signalling conditions:

  - At specific time intervals.
  - A delivery pipe goes from empty to not empty.
  - A delivery pipe goes from not full to full.
  - A delivery pipe goes from not empty to empty.
  - A delivery pipe goes from full to not full.
  - A control element is placed in a delivery pipe.
  - A control element is removed from a delivery pipe.

- Surrogate offset information

  The offsets in the Surrogate Control areas (surrogate start-of-free and surrogate start-of-element) are used by both the enqueue and dequeue logics.

- Surrogate status information

  Information regarding the current state of the delivery pipes (full, empty, wrap, etc.) is maintained for both pipes.

- Timer information

  A watchdog or idle timer can be used to ensure reliable operation of the delivery level in case interrupts are masked or lost. If the timer is used, both units must support it and be aware of the frequency used and the source.

Peer Level

Information from the peer-level configuration is used by the system management to identify adapter-to-adapter pairings. This information is needed during peer-level initialization to provide each of the adapters with the parameters it needs to carry out unit-level initialization with its peers.

Initialization

Unit Level Initialization

Units are initialized by the system unit at power-on. The POST (power-on self-test) performs the initialization using the setup information maintained in non-volatile RAM.

System Level Initialization

System-level initialization is carried out by a system unit at IPL time and by an adapter at IML time. The system-level initialization is a multistage process. For system unit to adapter pairs, the first stage is carried out by the system unit, the second stage by the adapter, and the third stage by the system unit. In the third stage, the system unit verifies that initialization is complete and notifies the adapter of the status.

For adapter-to-adapter pairs, the initialization is done under the control of system management as a third party operation, using the initialization control element.

The first stage of the initialization is done by the system unit support when it is loaded. As part of its initialization process, this support constructs a configuration record for each bus unit it supports. The configuration record is constructed in shared memory and its address passed to each adapter during unit-level initialization. The format and content of the configuration record is the same for all adapters and is defined in Figure 5-1 on page 5-5.

During system-level initialization, the term *out-bound* refers to the direction of flow and identifies the delivery pipe used to pass control elements from the system unit to the adapter. The term *in-bound* refers to the direction of flow and identifies the delivery pipe used to pass control elements from the adapter to the system unit.

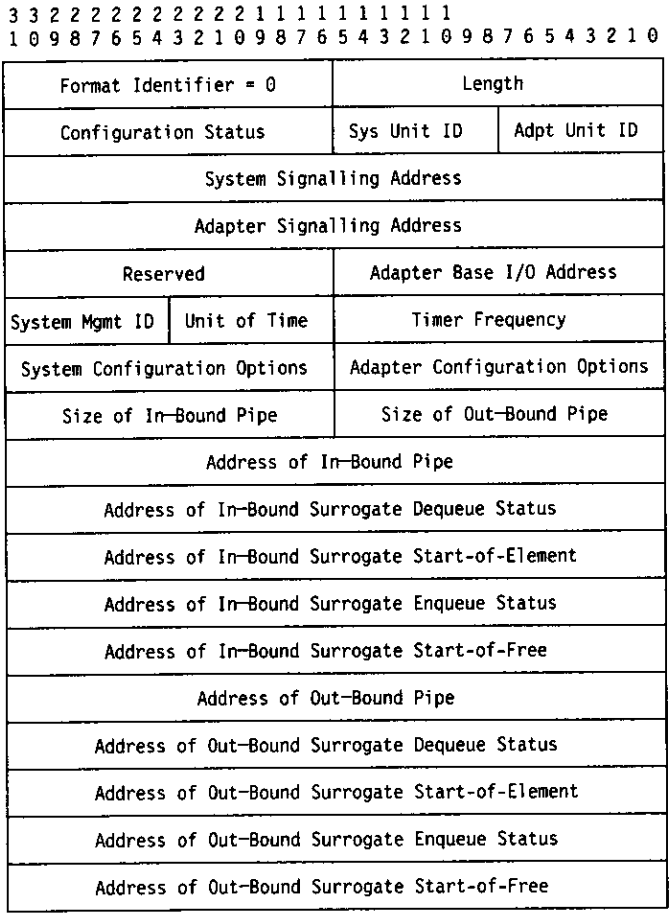

Figure 5-1. System Configuration Record Format

Format Identifier Field

This field identifies the format of the configuration record. Configuration records with a Format Identifier field value of 0 indicate a Move mode configuration record. All other values are reserved.

Length Field

This field contains the length of the configuration record in bytes, including the two bytes required for the Length field.

Configuration Status Field

This field holds status information that shows the current state of the initialization process. It is set to all 1s by the system unit after the configuration record is built.

System and Adapter Unit Identification Fields

These fields are set by the system unit and indicate the unit assignments (logical to physical mapping) as well as the unit pairing.

System Signalling Address Field

This field identifies the shared memory location of the Signalling Control area where the adapter is to place signalling information prior to signalling the system unit. The format and content of the Signalling Control area can be found in Figure 1-20 on page 1-30.

Adapter Signalling Address Field

This field identifies the shared memory location of the Signalling Control area where the system unit should place signalling information prior to signalling the adapter. The format and content of the Signalling Control area can be found in Figure 1-20 on page 1-30.

Adapter Base I/O Address Field

This field contains the base I/O address assigned to the adapter during Setup.

System Management ID Field

This field contains the entity identifier of the system-management entity in the system unit.

Unit of Time Field

This field indicates the granularity of the value in the Timer Value field. It is interpreted as:

01h - milliseconds
02h - 10s of milliseconds
04h - 100s of milliseconds
08h - seconds
10h - minutes

All other values are reserved.

Timer Frequency Field

The value in this field, when used with the Unit of Time field, indicates the frequency the system unit will present a timer-expiration signal of the paired unit. This field is valid only if the signal-on-timer-expiration bit is also set in the Configuration Options field of the appropriate bus unit.

System Configuration Options Field

This field contains information describing the system unit configuration options. The format and content of the field are found in the following figure.

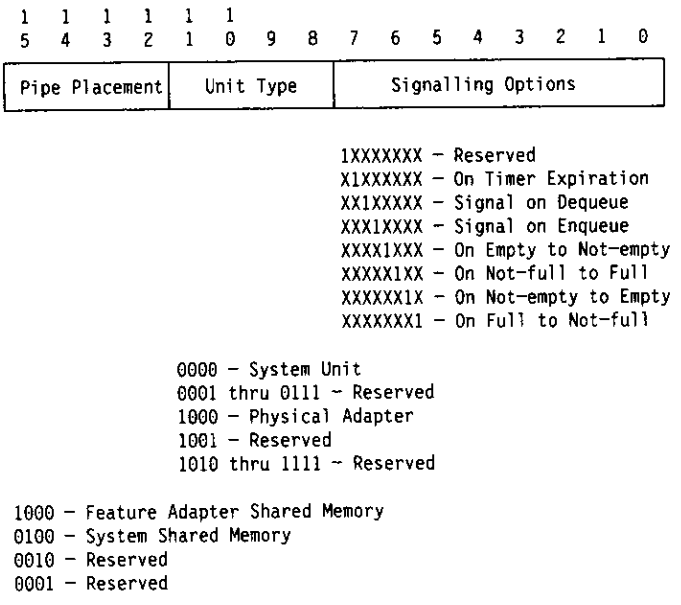

Figure 5-2. Configuration Field Format

Adapter Configuration Options Field

This field contains information describing the adapter configuration options. The format and content of the field are found in Figure 5-2 on page 5-7.

Size of In-Bound and Size of Out-Bound Pipe Fields

These fields define the amount of storage in bytes that is allocated for each pipe.

Note: For details on the roles of the remaining fields in the configuration record, refer to Figure 3-4 on page 3-9. The definitions for the pipe related control areas are found in "Delivery Pipe" on page 1-19.

Address of In-Bound Pipe Field

This field contains the address, in shared memory, for the pipe from the adapter to the system unit.

Address of In-Bound Surrogate Dequeue Status

This address contains the address, in shared memory, of the Surrogate Dequeue Status Control area that is written by the dequeue logic in the adapter and read by the enqueue logic in the system unit.

Address of In-Bound Surrogate Start-of-Element

This field contains the address, in shared memory, of the Surrogate Start of Element Control area which is written by the dequeue logic in the adapter and read by the enqueue logic in the system unit.

Address of In-Bound Surrogate Enqueue Status

This field contains the address, in shared memory, of the Surrogate Enqueue Status Control area that is written by the enqueue logic and read by the dequeue logic.

Address of In-Bound Surrogate Start-of-Free

This field contains the address, in shared memory, of the Surrogate Start of Free Control area that is written by the enqueue logic and read by the dequeue logic.

Address of Out-Bound Pipe Field

This field contains the base address, in shared memory, of the delivery pipe used to pass control elements from the system unit to the adapter.

Address of Out-Bound Surrogate Dequeue Status

This field contains the address, in shared memory, of the Surrogate Dequeue Status Control area that is written by the dequeue logic and read by the enqueue logic.

Address of Out-Bound Surrogate Start-of-Element

This field contains the address, in shared memory, of the Surrogate Start of Element Control area that is written by the dequeue logic and read by the enqueue logic.

Address of Out-Bound Surrogate Enqueue Status

This field contains the address, in shared memory, of the Surrogate Enqueue Status Control area that is written by the enqueue logic and read by the dequeue logic.

Address of Out-Bound Surrogate Start-of-Free

This field contains the address, in shared memory, of the Surrogate Start of Free Control area that is written by the enqueue logic and read by the dequeue logic.

The information required by system-level initialization can be obtained from any or all of the following places:

- Adapter description files and programs (ADF/ADP)
- Parameters provided at device-driver installation
- A separate configuration file created and maintained specifically for initialization
- Information compiled or assembled directly into the device driver itself.

In general, system-level initialization is concerned with the information required and not with how the information is obtained.

Peer Level Initialization

Peer-level, or adapter-to-adapter initialization, is performed following system-level initialization under the control of the system manager.

The configuration record used in peer-to-peer initialization is passed in an initialization request control element between the system-management entity in the system unit and the unit-management entity in the adapter (Figure 5-3 on page 5-11).

In the following figure, the term *Your* represents the unit that is the target of the initialization. The term *Peer* represents the unit that is the source of the initialization.

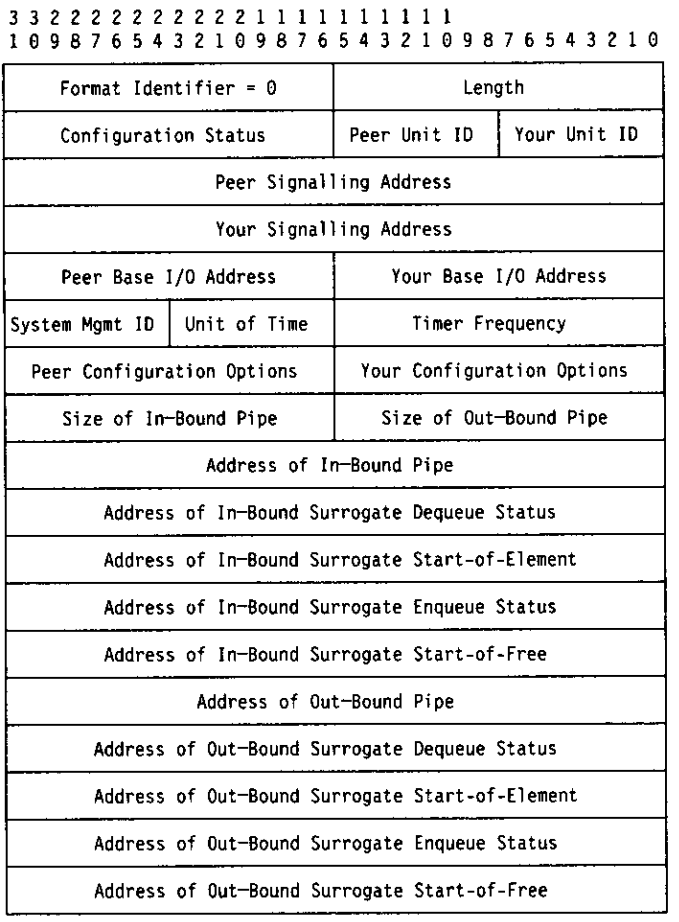

Figure 5-3. Peer Configuration Record Format

Peer-level initialization is similar to system-level initialization, except that it is performed only for the adapter-to-adapter pairs, and is performed using the management entities in the affected units. System-level initialization must be complete prior to peer-level initialization. Basically, the system-management entity acts as the control point for peer-level initialization. Its function is:

1. Building an initial peer-configuration record for each of the peer adapters.
2. Sending the configuration record to the management entity (entity 0) in the first adapter of the pair for processing. This is

accomplished by placing the configuration record into an initialization request control element.

3. Waiting for a response to the initialization request from the management entity in the first adapter of the pair.

   Upon receiving the initialization request, the management entity in the adapter examines and optionally indicates changes to the configuration record. Then, it returns the configuration record to the system-management entity in the system unit using either a reply or error control element.

4. After receiving a reply, the system-management entity sends the configuration record with the first unit's information to the management entity in the second adapter for processing. This is also accomplished by placing the configuration record into an initialization request control element.

   The second unit follows the same procedure as the first unit and returns either a reply or error control element to the system-management entity after indicating any needed changes to the configuration record.

5. After receiving a reply control element with changes to the configuration record, the system manager will reissue the initialization request to the unit-management entity in the first adapter, completing the initialization and confirming the establishment of its delivery pipes.

6. The system-management entity notifies the unit-management entity in the second adapter that the delivery pipes are established and that the flow of information between the peer adapters can now begin.

Any failure during the peer-level-initialization process is reported by an error control element and is processed by the system-management entity.

The manner in which adapters or system units are initially loaded (IPLed) is not defined by the architecture.

Exception Handling

At the delivery level, certain delivery-related exception conditions must be handled by the enqueue logic and dequeue logic. If the following exceptions are detected during the specified operation, the following action will be taken.

- Enqueue of a control element
  - Delivery pipe is full - return an indication to the caller.
  - Destination unit does not exist - return an indication to the caller.
- Dequeue of a control element
  - Delivery pipe is empty - return an indication to the caller.
  - Destination entity does not exist - the action taken is dependent on the send or receive interface policy. The control element containing the error can be discarded or returned to the originator with the appropriate exception indication noted in the Status field of the error control element.
  - Destination entity not active - the action taken is dependent on the send or receive interface policy. The control element can be discarded, returned to the originator, or be queued for retrieval when the entity becomes active.

At the physical level, the following exception conditions must be detected and reported so that recovery actions can be taken by the affected units (system unit and adapter):

- Channel data and address parity exceptions
- Channel non-parity exceptions
- Master-dependent and slave-dependent exceptions
- Castrophic exceptions
- Channel-timeout exceptions.

The recovery action taken is based on the operation in progress at the time of the exception. Exceptions during the following operations will result in the indicated recovery actions:

- Memory read or write of a control element

- Retry the read or write of the control element until a specified threshold value has been reached.
- Notify the unit-management entity for possible higher-level recovery by the system manager.

- Memory read or write of the information in the Enqueue or Dequeue Control areas
  - Notify the unit-management entity for possible higher-level recovery by the system manager.
- Memory read or write of the information in the Signalling Control areas
  - Retry the read or write operation until the a specified threshold value has been reached.
  - Notify the unit-management entity for possible higher-level recovery by the system manager.
- I/O read or write of a port in I/O address space
  - Retry the read or write operation until the a specified threshold value has been reached.
  - Notify the unit-management entity for possible higher-level recovery by the system manager.
- Memory read or write of data
  - Retry the read or write operation until the a specified threshold value has been reached.
  - Retry the command in the control element which initiated the data read or write operation.
  - Notify the unit management entity for possible higher-level recovery by the system manager.

Management Relationships

The Move mode architecture also defines the relationship of the various parts of the delivery services to the unit-management services.

The unit-management services can be viewed as the platform on

which future delivery-management support and RAS-type services for bus masters are built.

The overall relationships for unit-level management are shown in Figure 5-4

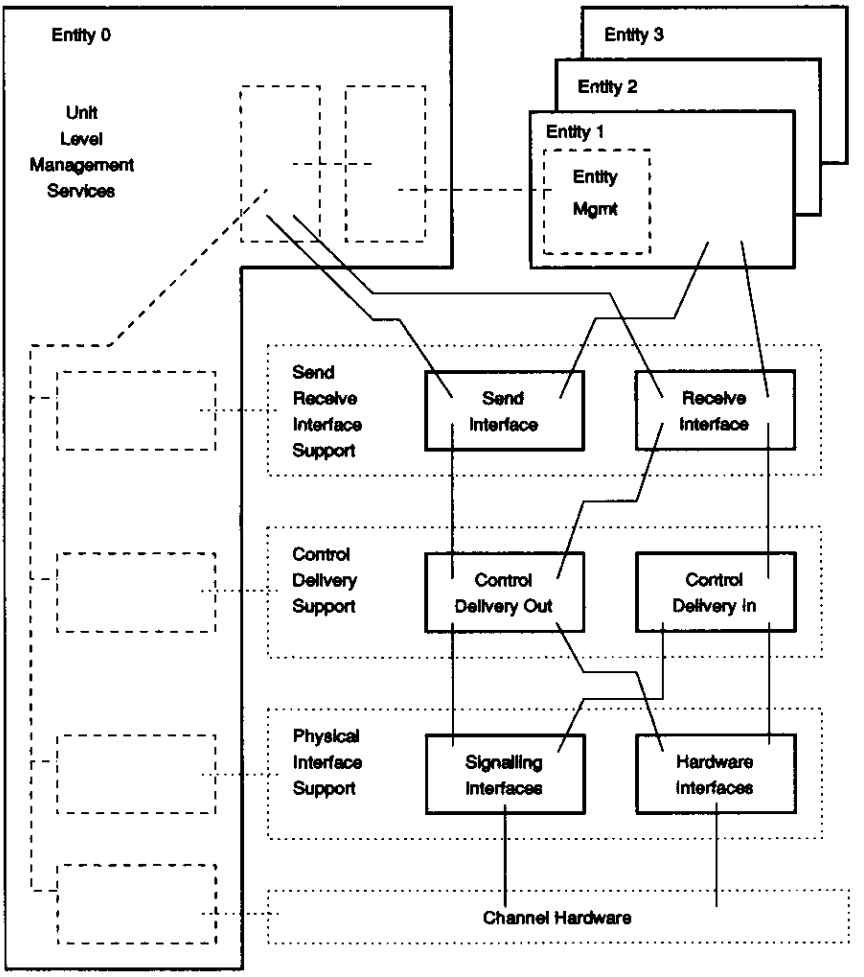

Figure 5-4. Unit Management Relationships

There are different forms of relationships between the various parts of the Move mode support and the unit-level management services.

- The management services are provided by a management entity that is defined as entity 0. There is an entity 0 in each unit in the configuration. This allows the overall management services to send and receive management-type control elements to and from the various units in the configuration after the delivery support is initialized.

  There can be control elements:

  - To reset the delivery pipes
  - To notify the overall-management entity of events in the unit
  - To report detail-error information for entities within the unit
  - To run tests on the unit.

- Unit management services need independent operations of entity-to-entity delivery and are required to have an independent unit of execution (thread).

- There is a portion of unit-management services that can be used as logic that actually handles the turn-around for requests with delivery-type errors. It also provides the coupling between the receive-interface logic and the enqueue logic for the pipe in the opposite direction.

- There can be other portions of management support to handle entity-level and physical-level error conditions as well as other management-related operations.

- Initialization can also be considered part of unit-management services.

The architecture requires that entity 0 be used for management-type services and not for client-type or server-type entities.

Chapter 6. Architectural Compliance

The architecture defines the Move mode protocols used to deliver control elements between pairs of functional entities located in different system units or adapters. Several functional entities can exist within a system unit or adapter. Different entities within a specific system unit can be located in different system units or adapters.

By complying with the architecture, two compliant system units or adapters, that are interconnected by a physical media and share addressable memory, can support the same base set of delivery protocols between entity pairs.

The architecture logically structures the delivery service into three functional layers:

- The physical level
- The delivery level
- The processing level.

The statements of compliance for each level define the requirements and the portion which can be configured. In addition to the protocols defined as base requirements, there are other optional architected protocols for a given implementation.

Physical Level

The architecture defines several protocols for the physical-level primitives.

Base Protocols

To comply with the architecture, a product must:

- Implement the architected physical-level protocols.
- Maintain compatibility with the existing products by using an adapter that can use the physical level to support private protocols.

Additional Architected Protocols

Because the initialization protocol is product defined, the implementation of a physical-level protocol might not implement any port which is not used as part of the physical-level delivery logic.

Delivery Level

The delivery-level protocols include the signalling protocol, the enqueue protocol, the dequeue protocol, and the receive-interface protocol.

Base Protocols

To comply with the architecture, a product must:

- Implement the signalling protocol for the six state-change signals defined by the signalling logic in "Enqueue Protocol" on page 3-11, and "Dequeue Protocol" on page 3-19.
- Implement Signalling Control areas for each unit as defined in "Signalling Control Area" on page 1-29.
- Implement the delivery pipes for each pair of units supported as defined by the above references and Chapter 3, "Delivery Level" on page 3-1.

- Implement the delivery-pipe structure defined in "Delivery Pipe" on page 1-19.
- Implement the function defined for control element enqueue in "Enqueue Protocol" on page 3-11 and the Enqueue Surrogate Control areas defined in "Surrogate Enqueue Control Area" on page 1-24.
- Implement the function defined for control element dequeue in "Dequeue Protocol" on page 3-19 and the Dequeue Surrogate Control areas defined in "Surrogate Dequeue Control Area" on page 1-28.

Additional Architected Protocols

The architecture defines the signalling of state-changes based on device-dependent intervals for product implementation.

The architecture defines a management signal to indicate that initialization is complete. The support for this during initialization is optional for a product implementation.

The architecture defines receive-interface error handling which is optional for a product implementation.

Processing Level

In general, the entity-to-entity protocols are not defined by the architecture. However, to comply with the architecture, certain portions of the control element are supported by the delivery level.

Base Protocols

To comply with the architecture, a product must:

- Implement the control element structure, defined in "Control Elements" on page 1-32, on all control elements used.

  This header contains the following fields:

  - Format Identifier
  - Length
  - Common Indicators

- Source Unit
- Source Entity
- Destination Unit
- Destination Entity
- Correlation ID.

- Implement the bits of the Common Indicators field if these functions are used.
- Implement the architected form of the control element when the same functions are used.

Additional Architected Protocols

The architecture defines a number of common-use control elements which are optional.

Management

The architecture defines protocols for configuring the delivery services and for initializing each unit (system unit and adapter) that will use the delivery service.

Base Protocols

To comply with the architecture, a product must :

- Implement the configuration record as defined in "Configuration" on page 5-1.
- Implement entity 0 as a management entity.

Additional Architected Protocols

The protocols for using the configuration record are device-dependent.

Appendix A. C Language

```
/*
 +-------------------------------------------------------------------+
 | Header File: MOVMODE.H                                            |
 |                                                                   |
 | Description: Contains the include files, constants, data structures,|
 |              macro definitions, and function declarations common to |
 |              all Move mode Delivery Services components.          |
 |                                                                   |
 | Status Information:                                               |
 |                                                                   |
 |     Created: 04/01/89                                             |
 |     Revised: 01/10/90                                             |
 |     Revised: 07/17/90                                             |
 +-------------------------------------------------------------------+
*/
#ifndef _MOVMODE_H
#define _MOVMODE-H
/*
                +---------------------------------------+
                |        USEFUL TYPE DEFINITIONS        |
                +---------------------------------------+
*/
typedef int                 BOOLEAN;
typedef unsigned char       BYTE;       /* full 8 bit char           */
typedef unsigned char      *pBYTE;      /* pointer to 8 bit char     */
typedef unsigned char  far *fpBYTE;     /* far pointer to 8 bit char */
typedef unsigned short      WORD;       /* full 16 bit word          */
typedef unsigned short     *pWORD;      /* pointer to 16 bit word    */
typedef unsigned short far *fpWORD;     /* far pointer to 16 bit word */
typedef unsigned long       DWORD;      /* full 32 bit word          */
typedef unsigned long      *pDWORD;     /* pointer to 32 bit word    */
typedef unsigned long  far *fpDWORD;    /* far pointer to 32 bit word */

/*
                +---------------------------------------+
                |             INCLUDE FILES             |
                +---------------------------------------+
*/
#include <stdio.h>              /* standard stream I/O               */

#ifdef __TURBOC__
   #include <alloc.h>           /* Turbo C memory allocation         */
   #include <mem.h>             /* memory manipulation               */
#else
   #include <malloc.h>          /* MS/IBM C memory allocation        */
   #include <memory.h>          /* memory manipulation               */
#endif

#include <stdlib.h>             /* standard library                  */
#include <ctype.h>              /* char classification/conversion    */

#include "element.h"            /* Control elements                  */
#include "pipe.h"               /* Pipe (delivery queues)            */
```

```
/*
                +---------------------------------------+
                |          DEFINED CONSTANTS            |
                +---------------------------------------+
*/
#define FALSE   0
#define TRUE    1

#define OFF     0
#define ON      1

#define SUCCESS 1
#define FAILURE 0

#endif
```

```
/*
 +-------------------------------------------------------------------+
 | Header File: pipe.h                                               |
 |                                                                   |
 | Description: Contains the data structure and definitions for the  |
 |              following control areas in shared memory address     |
 |              space:                                               |
 |               - configuration record                              |
 |               - local enqueue control area                        |
 |               - surrogate enqueue control areas                   |
 |               - local dequeue control area                        |
 |               - surrogate dequeue control areas                   |
 |               - signalling areas                                  |
 |                                                                   |
 | Status Information:                                               |
 |                                                                   |
 |     Created: 05/02/89                                             |
 |     Revised: 01/10/90                                             |
 |     Revised: 07/17/90                                             |
 +-------------------------------------------------------------------+
*/
#ifndef _PIPE_H
#define _PIPE_H
/*
                +--------------------------------------+
                |           DATA STRUCTURES            |
                +--------------------------------------+
*/

/* -----
    The flags within the Configuration Record are used to indicate
    the specific options to be used between a pair of bus units.
                                                          ----- */

#ifdef __TURBOC__

typedef struct {                /* Configuration Options            */
                                /*  Physical Placement of the Pipe  */
  unsigned adp_mem    : 1;      /*  - bit 15 - in adapter           */
                                /*    0 = outbound pipe in adapter  */
                                /*    1 = inbound pipe in adapter   */
  unsigned sys_mem    : 1;      /*  - bit 14 - in system unit       */
                                /*    0 = outbound pipe in system   */
                                /*    1 = inbound pipe in system    */
  unsigned b13        : 1;      /*  - bit 13 - reserved             */
  unsigned b12        : 1;      /*  - bit 12 - reserved             */
  unsigned unit_type  : 4;      /*  Bus Unit Type                   */
                                /*  - 0000 - system unit            */
                                /*  - 0001 thru 0111 - reserved     */
                                /*  - 1000 - adapter type 1         */
                                /*  - 1001 - adapter type 2         */
                                /*  - 1010 thru 1111 - reserved     */
                                /*  Signalling Conditions           */
  unsigned b7         : 1;      /*  - bit 7  - reserved             */
  unsigned on_timer   : 1;      /*  - bit 6  - on timer expiration  */
  unsigned on_notfull : 1;      /*  - bit 5  - on full to non-full  */
  unsigned on_deque   : 1;      /*  - bit 4  - on dequeue element   */
  unsigned on_enque   : 1;      /*  - bit 3  - on enqueue element   */
```

```
  unsigned on_empty   : 1;      /*  - bit 2  - on pipe empty          */
  unsigned on_full    : 1;      /*  - bit 1  - on pipe full           */
  unsigned on_notempty: 1;      /*  - bit 0  - on empty to non-empty  */
} CFG_OPT;

#else

typedef struct {                /* Configuration Options              */
                                /*  Signalling Conditions             */
  unsigned on_notempty: 1;      /*  - bit 0  - on empty to non-empty  */
  unsigned on_full    : 1;      /*  - bit 1  - on pipe full           */
  unsigned on_empty   : 1;      /*  - bit 2  - on pipe empty          */
  unsigned on_enque   : 1;      /*  - bit 3  - on enqueue element     */
  unsigned on_deque   : 1;      /*  - bit 4  - on dequeue element     */
  unsigned on_notfull : 1;      /*  - bit 5  - on full to non-full    */
  unsigned on_timer   : 1;      /*  - bit 6  - on timer expiration    */
  unsigned b7         : 1;      /*  - bit 7  - reserved               */
  unsigned unit_type  : 4;      /*  Bus Unit Type                     */
                                /*  - 0000 - system unit              */
                                /*  - 0001 thru 0111 - reserved       */
                                /*  - 1000 - adapter type 1           */
                                /*  - 1001 - adapter type 2           */
                                /*  - 1010 thru 1111 - reserved       */
  unsigned b12        : 1;      /*  - bit 12 - reserved               */
  unsigned b13        : 1;      /*  - bit 13 - reserved               */
                                /*  Physical Placement of the Pipe    */
  unsigned sys_mem    : 1;      /*  - bit 14 - in system unit         */
                                /*    0 = outbound pipe in system     */
                                /*    1 = inbound pipe in system      */
  unsigned adp_mem    : 1;      /*  - bit 15 - in adapter             */
                                /*    0 = outbound pipe in adapter    */
                                /*    1 = inbound pipe in adapter     */
} CFG_OPT;

#endif

/* -----
    The fields within the Configuration Record (CFG_RCD) are used
    at initialization time to identify the location of control areas
    in shared memory address space and the options to be used when
    using the delivery service between a pair of bus units.
                                                              ----- */

typedef struct {                /* Configuration Record               */
  WORD    fid;                  /* - Format Identifier                */
  WORD    len;                  /* - length field                     */
  WORD    status;               /* - configuration status field       */
  BYTE    sid;                  /* - system unit identification field */
  BYTE    aid;                  /* - adapter identification field     */
  fpDWORD ss_addr;              /* - system signalling address field  */
  fpDWORD as_addr;              /* - adapter signalling address field */
  WORD    peer_io_addr;         /* - base I/O addr - peer (from)      */
  WORD    base_io_addr;         /* - base I/O addr - adapter (to)     */
  BYTE    smgmt_eid;            /* - system manager entity id field   */
  BYTE    time_units;           /* - units of time                    */
  WORD    time_freq;            /* - timer frequency                  */
  CFG_OPT sys_config;           /* - system unit configuration options*/
  CFG_OPT adp_config;           /* - adapter configuration options    */
```

```
  WORD    out_size;           /* - out-bound pipe size (sys to adp) */
  WORD    in_size;            /* - in-bound pipe size (adp to sys)  */
  fpDWORD in_pipe;            /* - in-bound pipe address            */
  fpDWORD in_sds;             /* - in-bound surrogate dequeue       */
                              /*     status area address            */
  fpDWORD in_sse;             /* - in-bound surrogate start of      */
                              /*     elements control area          */
  fpDWORD in_ses;             /* - in-bound surrogate enqueue       */
                              /*     status area address            */
  fpDWORD in_ssf;             /* - in-bound surrogate start of      */
                              /*     free control area address      */
  fpDWORD out_pipe;           /* - out-bound pipe address           */
  fpDWORD out_sds;            /* - out-bound surrogate dequeue      */
                              /*     status area address            */
  fpDWORD out_sse;            /* - out-bound surrogate start of     */
                              /*     elements control area address */
  fpDWORD out_ses;            /* - out-bound surrogate enqueue      */
                              /*     status area address            */
  fpDWORD out_ssf;            /* - out-bound surrogate start of     */
                              /*     free control area address      */
} CFG_RCD;

/* -----
    The flags within the Enqueue Status Word are used by the Enqueue
    Logic to maintain local state information at the enqueue end of
    the pipe.
                                                                ----- */
#ifdef __TURBOC__

typedef struct {              /* Enqueue Status Word Flags          */
  unsigned b15       : 1;     /*  - bit 15 - reserved               */
  unsigned b14       : 1;     /*  - bit 14 - reserved               */
  unsigned b13       : 1;     /*  - bit 13 - reserved               */
  unsigned b12       : 1;     /*  - bit 12 - reserved               */
  unsigned queued    : 1;     /*  - bit 11 - element stored         */
  unsigned b10       : 1;     /*  - bit 10 - reserved               */
  unsigned b9        : 1;     /*  - bit 9  - reserved               */
  unsigned wrap      : 1;     /*  - bit 8  - enqueue wrap           */
  unsigned b7        : 1;     /*  - bit 7  - reserved               */
  unsigned b6        : 1;     /*  - bit 6  - reserved               */
  unsigned b5        : 1;     /*  - bit 5  - reserved               */
  unsigned b4        : 1;     /*  - bit 4  - reserved               */
  unsigned b3        : 1;     /*  - bit 3  - reserved               */
  unsigned b2        : 1;     /*  - bit 2  - reserved               */
  unsigned empty     : 1;     /*  - bit 1  - empty                  */
  unsigned full      : 1;     /*  - bit 0  - queue full             */
} LES_FLAGS;

#else

typedef struct {              /* Enqueue Status Word Flags          */
  unsigned full      : 1;     /*  - bit 0  - queue full             */
  unsigned empty     : 1;     /*  - bit 1  - empty                  */
  unsigned b2        : 1;     /*  - bit 2  - reserved               */
  unsigned b3        : 1;     /*  - bit 3  - reserved               */
  unsigned b4        : 1;     /*  - bit 4  - reserved               */
  unsigned b5        : 1;     /*  - bit 5  - reserved               */
  unsigned b6        : 1;     /*  - bit 6  - reserved               */
```

```
  unsigned b7         : 1;      /*  - bit 7  - reserved              */
  unsigned wrap       : 1;      /*  - bit 8  - enqueue wrap          */
  unsigned b9         : 1;      /*  - bit 9  - reserved              */
  unsigned b10        : 1;      /*  - bit 10 - reserved              */
  unsigned queued     : 1;      /*  - bit 11 - element stored        */
  unsigned b12        : 1;      /*  - bit 12 - reserved              */
  unsigned b13        : 1;      /*  - bit 13 - reserved              */
  unsigned b14        : 1;      /*  - bit 14 - reserved              */
  unsigned b15        : 1;      /*  - bit 15 - reserved              */
} LES_FLAGS;

#endif

/* -----
    The fields within the Local Enqueue Control Area (LECA) identify
    the location of the pipe, indicate the status of the pipe, and
    provide state information identifying the starting and ending
    offsets of control elements within the pipe.
                                                                ----- */

typedef struct {                /* Local Enqueue Control Area       */
  fpDWORD base;                 /* - Pipe Address                   */
  WORD   we;                    /* - offset to wrap element         */
  union {                       /* - enqueue status                 */
     WORD      fld;             /*   as a word                      */
     LES_FLAGS flg;             /*   as individual flags            */
  } es;
  WORD   sf;                    /* - start of free space in pipe    */
  WORD   ef;                    /* - end of free space in pipe      */
  WORD   end;                   /* - offset to end (of queue space) */
  WORD   top;                   /* - offset to top (wrap point)     */
} LECA;

/* -----
    The flags within the Surrogate Enqueue Status Word are used by
    the Enqueue Logic to convey the state of the pipe to the Dequeue
    Logic at the other end of the pipe.
                                                                ----- */
#ifdef __TURBOC__

typedef struct {                /* Surrogate Enqueue Status Word Flag */
  unsigned b15        : 1;      /*  - bit 15 - reserved             */
  unsigned b14        : 1;      /*  - bit 14 - reserved             */
  unsigned b13        : 1;      /*  - bit 13 - reserved             */
  unsigned b12        : 1;      /*  - bit 12 - reserved             */
  unsigned b11        : 1;      /*  - bit 11 - reserved             */
  unsigned b10        : 1;      /*  - bit 10 - reserved             */
  unsigned b9         : 1;      /*  - bit 9  - reserved             */
  unsigned wrap       : 1;      /*  - bit 8  - wrap of pipe occurred */
  unsigned b7         : 1;      /*  - bit 7  - reserved             */
  unsigned b6         : 1;      /*  - bit 6  - reserved             */
  unsigned b5         : 1;      /*  - bit 5  - reserved             */
  unsigned b4         : 1;      /*  - bit 4  - reserved             */
  unsigned b3         : 1;      /*  - bit 3  - reserved             */
  unsigned b2         : 1;      /*  - bit 2  - reserved             */
  unsigned b1         : 1;      /*  - bit 1  - reserved             */
  unsigned full       : 1;      /*  - bit 0  - pipe is full         */
} SES_FLAGS;
```

```
#else

typedef struct {                    /* Surrogate Enqueue Status Word Flag */
  unsigned full       : 1;          /*  - bit 0  - pipe is full           */
  unsigned b1         : 1;          /*  - bit 1  - reserved               */
  unsigned b2         : 1;          /*  - bit 2  - reserved               */
  unsigned b3         : 1;          /*  - bit 3  - reserved               */
  unsigned b4         : 1;          /*  - bit 4  - reserved               */
  unsigned b5         : 1;          /*  - bit 5  - reserved               */
  unsigned b6         : 1;          /*  - bit 6  - reserved               */
  unsigned b7         : 1;          /*  - bit 7  - reserved               */
  unsigned wrap       : 1;          /*  - bit 8  - wrap of pipe occurred  */
  unsigned b9         : 1;          /*  - bit 9  - reserved               */
  unsigned b10        : 1;          /*  - bit 10 - reserved               */
  unsigned b11        : 1;          /*  - bit 11 - reserved               */
  unsigned b12        : 1;          /*  - bit 12 - reserved               */
  unsigned b13        : 1;          /*  - bit 13 - reserved               */
  unsigned b14        : 1;          /*  - bit 14 - reserved               */
  unsigned b15        : 1;          /*  - bit 15 - reserved               */
} SES_FLAGS;

#endif

typedef struct {                    /* Surrogate Enqueue Control Area     */
  union {                           /* - surrogate enqueue status         */
     WORD      fld;                 /*   as a word                        */
     SES_FLAGS flg;                 /*   as individual flags              */
  } ses;
  WORD  ssf;                        /* - offset to start of free space    */
  } SECA;

/* -----
    The flags within the Dequeue Status Word are used by the Dequeue
    Logic to maintain local state information at the dequeue end of
    pipe.
                                                                  ----- */

#ifdef __TURBOC__

typedef struct {                    /* Dequeue Status Word Flags          */
  unsigned b15        : 1;          /*  - bit 15 - reserved               */
  unsigned b14        : 1;          /*  - bit 14 - reserved               */
  unsigned b13        : 1;          /*  - bit 13 - reserved               */
  unsigned b12        : 1;          /*  - bit 12 - reserved               */
  unsigned dequeued   : 1;          /*  - bit 11 - element removed        */
  unsigned prempt     : 1;          /*  - bit 10 - stop dequeue operation */
  unsigned b9         : 1;          /*  - bit 9  - reserved               */
  unsigned wrap       : 1;          /*  - bit 8  - wrap element toggle    */
  unsigned b7         : 1;          /*  - bit 7  - reserved               */
  unsigned b6         : 1;          /*  - bit 6  - reserved               */
  unsigned b5         : 1;          /*  - bit 5  - reserved               */
  unsigned b4         : 1;          /*  - bit 4  - reserved               */
  unsigned b3         : 1;          /*  - bit 3  - reserved               */
  unsigned b2         : 1;          /*  - bit 2  - reserved               */
  unsigned empty      : 1;          /*  - bit 1  - pipe is empty          */
  unsigned full       : 1;          /*  - bit 0  - full                   */
} LDS_FLAGS;
```

```
#else

typedef struct {                    /* Dequeue Status Word Flags          */
  unsigned full       : 1;          /*  - bit 0  - full                   */
  unsigned empty      : 1;          /*  - bit 1  - pipe is empty          */
  unsigned b2         : 1;          /*  - bit 2  - reserved               */
  unsigned b3         : 1;          /*  - bit 3  - reserved               */
  unsigned b4         : 1;          /*  - bit 4  - reserved               */
  unsigned b5         : 1;          /*  - bit 5  - reserved               */
  unsigned b6         : 1;          /*  - bit 6  - reserved               */
  unsigned b7         : 1;          /*  - bit 7  - reserved               */
  unsigned wrap       : 1;          /*  - bit 8  - wrap element toggle    */
  unsigned b9         : 1;          /*  - bit 9  - reserved               */
  unsigned prempt     : 1;          /*  - bit 10 - stop dequeue operation */
  unsigned dequeued   : 1;          /*  - bit 11 - element removed        */
  unsigned b12        : 1;          /*  - bit 12 - reserved               */
  unsigned b13        : 1;          /*  - bit 13 - reserved               */
  unsigned b14        : 1;          /*  - bit 14 - reserved               */
  unsigned b15        : 1;          /*  - bit 15 - reserved               */
} LDS_FLAGS;

#endif

/* -----
    The fields within the Local Dequeue Control Area (LDCA) identify
    the location of the pipe, indicate the status of the pipe, and
    provide state information identifying the starting and ending
    offsets of control elements within the pipe.
                                                                  ----- */

typedef struct {                    /* Local Dequeue Control Area         */
  fpDWORD base;                     /* - Pipe Address                     */
  WORD   we;                        /* - offset to wrap element           */
  union {                           /* - Dequeue status                   */
     WORD      fld;                 /*   as a word                        */
     LDS_FLAGS flg;                 /*   as individual flags              */
  } ds;
  WORD   se;                        /* - start of elements in pipe        */
  WORD   ee;                        /* - end of elements in pipe          */
  WORD   end;                       /* - offset to end (of queue space)   */
  WORD   top;                       /* - offset to top (wrap point)       */
} LDCA;

/* -----
    The flags within the Surrogate Dequeue Status Word are used by
    the Dequeue Logic to convey the state of the pipe to the Enqueue
    Logic at the other end of the pipe.
                                                                  ----- */
#ifdef __TURBOC__

typedef struct {                    /* Surrogate Dequeue Status Flags     */
  unsigned b15        : 1;          /*  - bit 15 - reserved               */
  unsigned b14        : 1;          /*  - bit 14 - reserved               */
  unsigned b13        : 1;          /*  - bit 13 - reserved               */
  unsigned b12        : 1;          /*  - bit 12 - reserved               */
  unsigned b11        : 1;          /*  - bit 11 - reserved               */
  unsigned b10        : 1;          /*  - bit 10 - reserved               */
  unsigned b9         : 1;          /*  - bit 9  - reserved               */
```

```
  unsigned wrap       : 1;      /*  - bit 8  - wrap of pipe occurred  */
  unsigned b7         : 1;      /*  - bit 7  - reserved               */
  unsigned b6         : 1;      /*  - bit 6  - reserved               */
  unsigned b5         : 1;      /*  - bit 5  - reserved               */
  unsigned b4         : 1;      /*  - bit 4  - reserved               */
  unsigned b3         : 1;      /*  - bit 3  - reserved               */
  unsigned b2         : 1;      /*  - bit 2  - reserved               */
  unsigned empty      : 1;      /*  - bit 1  - pipe is empty          */
  unsigned b0         : 1;      /*  - bit 0  - reserved               */
} SDS_FLAGS;

#else

typedef struct {                /* Surrogate Dequeue Status Flags     */
  unsigned b0         : 1;      /*  - bit 0  - reserved               */
  unsigned empty      : 1;      /*  - bit 1  - pipe is empty          */
  unsigned b2         : 1;      /*  - bit 2  - reserved               */
  unsigned b3         : 1;      /*  - bit 3  - reserved               */
  unsigned b4         : 1;      /*  - bit 4  - reserved               */
  unsigned b5         : 1;      /*  - bit 5  - reserved               */
  unsigned b6         : 1;      /*  - bit 6  - reserved               */
  unsigned b7         : 1;      /*  - bit 7  - reserved               */
  unsigned wrap       : 1;      /*  - bit 8  - wrap of pipe occurred  */
  unsigned b9         : 1;      /*  - bit 9  - reserved               */
  unsigned b10        : 1;      /*  - bit 10 - reserved               */
  unsigned b11        : 1;      /*  - bit 11 - reserved               */
  unsigned b12        : 1;      /*  - bit 12 - reserved               */
  unsigned b13        : 1;      /*  - bit 13 - reserved               */
  unsigned b14        : 1;      /*  - bit 14 - reserved               */
  unsigned b15        : 1;      /*  - bit 15 - reserved               */
} SDS_FLAGS;

#endif

typedef struct {                /* Surrogate Dequeue Control Area     */
  union {                       /* - surrogate dequeue status         */
     WORD      fld;             /*   as a word                        */
     SDS_FLAGS flg;             /*   as individual flags              */
  } sds;
  WORD   sse;                   /* - offset to start of elements      */
} SDCA;

/* -----
    The fields within the Signalling Area (SIGNAL) are used to identify
    the source as well as the reason for signalling from one bus unit
    to another.
                                                                ----- */
typedef struct {                /* Signalling Control Area            */
  BYTE   nque;                  /*   - enqueue state change           */
  BYTE   mgmt;                  /*   - management state change        */
  BYTE   byte2;                 /*   - reserved                       */
  BYTE   dque;                  /*   - dequeue state change           */
} SCA;

#endif
```

```
/*
 +-------------------------------------------------------------------+
 | Header File: ELEMENT.H                                            |
 |                                                                   |
 | Description: Contains the data structure definitions for the basic |
 |              control elements.                                    |
 |                                                                   |
 | Status Information:                                               |
 |                                                                   |
 |     Created: 05/02/89                                             |
 |     Revised: 01/10/90                                             |
 |     Revised: 07/17/90                                             |
 +-------------------------------------------------------------------+
*/
#ifndef _ELEMENT_H
#define _ELEMENT_H
/*

               +--------------------------------------+
               |           DEFINED CONSTANTS          |
               +--------------------------------------+
*/
          /* ----- Format Identifier field values ----- */

#define FID_MM     0            /* SCB Move Mode Format */

      /* ----- Control Element Type field Element IDs ----- */

#define REQ_EL     0            /* request */
#define REP_EL     1            /* reply   */
#define EVT_EL     2            /* event   */
#define ERR_EL     3            /* error   */

#define ERR_EL_LEN 20           /* length of error element          */
#define DEL_EL_LEN 6            /* length of delivery wrap element  */

      /* ----- Control Element Type field Function Codes ----- */

                                /* 0 and 1 are reserved             */
#define FC_INIT      2          /* initialize                       */
                                /* 3 is reserved                    */
#define FC_READ      4          /* read                             */
#define FC_READL     5          /* read list                        */
#define FC_READI     6          /* read immediate                   */
#define FC_WRITE     7          /* write                            */
#define FC_WRITEL    8          /* write list                       */
#define FC_WRITEI    9          /* write immediate                  */
#define FC_EXL      10          /* execute list                     */
#define FC_MARK     11          /* mark                             */
#define FC_CANCEL   12          /* cancel                           */
#define FC_PARAM    13          /* parameters                       */
#define FC_RDCFG    14          /* read configuration               */
#define FC_DIAG     15          /* run diagnostics                  */
                                /* 16 is reserved                   */
#define FC_RESUME   17          /* resume (event)                   */
#define FC_NOTIFY   18          /* notification (event)             */
#define FC_INFORM   19          /* user status information (event)  */
```

```
                                /* 20 thru 62 are reserved              */
#define FC_WRAP     63          /* wrap of queue (internal)             */

      /* ----- Control Element Value field Error Codes ----- */

#define NO_RECV_PENDING 1

/*
                +--------------------------------------+
                |           DATA STRUCTURES            |
                +--------------------------------------+
*/
#ifdef __TURBOC__
typedef struct {                /* Common Indicators Field              */
  unsigned id       : 2;        /* - bit 15 - 14 identifier (ID)        */
  unsigned suppress : 1;        /* - bit 13 suppress reply flag (S)     */
  unsigned chain    : 2;        /* - bit 12 - 11 chaining flag (C)      */
  unsigned indirect : 1;        /* - bit 10 indirect flag  (I)          */
  unsigned notify   : 1;        /* - bit 9 notification flag  (N)       */
  unsigned wait     : 1;        /* - bit 8 wait flag (W)                */
  unsigned expedite : 1;        /* - bit 7 expedited flag (E)           */
  unsigned function : 7;        /* - bit 6 - 0 function code (FC)       */
} CIND_FLD;

#else

typedef struct {                /* Common Indicators Field              */
  unsigned function : 7;        /* - bit 6 - 0 function code (FC)       */
  unsigned expedite : 1;        /* - bit 7 expedited flag (E)           */
  unsigned wait     : 1;        /* - bit 8 wait flag (W)                */
  unsigned notify   : 1;        /* - bit 9 notification flag (N)        */
  unsigned indirect : 1;        /* - bit 10 indirect flag (I)           */
  unsigned chain    : 2;        /* - bit 12 - 11 chaining flag (C)      */
  unsigned suppress : 1;        /* - bit 13 suppress reply flag (S)     */
  unsigned id       : 2;        /* - bit 15 - 14 identifier (ID)        */
} CIND_FLD;

#endif

typedef struct {                /* Element Source/Destination Field     */
  BYTE eid;                     /* - entity identifier                  */
  BYTE uid;                     /* - unit identifier                    */
} ADDR_FLD;

typedef struct {                /* Element descriptor                   */
  WORD     fid;                 /* - Format Identifier                  */
  WORD     len;                 /* - length field                       */
  CIND_FLD cind;                /* - common indicators field            */
  WORD     res1;                /* - reserved for future use            */
  ADDR_FLD dest;                /* - destination field                  */
  ADDR_FLD srce;                /* - source field                       */
  DWORD    cor_id;              /* - correlation field                  */
  DWORD    val_fld;             /* - value field                        */
} ELEMENT;
#endif
```

Notes:

Appendix B. Assembler Language

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                                                                     *
;* INCLUDE FILE: MOVMODE.INC                                           *
;*                                                                     *
;*    COPYRIGHT: (C) Copyright IBM Corporation 1989, 1990.  All        *
;*               rights reserved.                                      *
;*                                                                     *
;*    REV LEVEL: 1.1                                                   *
;*                                                                     *
;*  DESCRIPTION: Contains constants, data structures, and macro        *
;*               definitions common to all Move Mode Delivery          *
;*               Services components.                                  *
;*                                                                     *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
INCLUDE "ELEMENT.INC"           ;control element
INCLUDE "PIPE.INC"              ;delivery pipes
```

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                                                                       *
;* INCLUDE FILE: PIPE.INC                                                *
;*                                                                       *
;*    COPYRIGHT: (C) Copyright IBM Corporation 1989, 1990.  All          *
;*               rights reserved.                                        *
;*                                                                       *
;*    REV LEVEL: 1.1                                                     *
;*                                                                       *
;*  DESCRIPTION: Contains data structure and definitions for the         *
;*               following control areas in shared memory address        *
;*               space:                                                  *
;*                                                                       *
;*               - configuration record                                  *
;*               - local enqueue control area                            *
;*               - surrogate enqueue control areas                       *
;*               - local dequeue control area                            *
;*               - surrogate dequeue control areas                       *
;*               - signalling areas                                      *
;*                                                                       *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

;*              ****************************************
;*              *           DATA STRUCTURES            *
;*              ****************************************

;*  Flags within the Configuration Word (CFG_WRD) are used to indicate
;*  specific options to be used between a pair of bus units.

CFG_OPT RECORD                  ;Physical Placement of the Pipe
        adp_mem:1,              ; - bit 15 - in feature adapter
                                ;   0 = outbound pipe in adapter
                                ;   1 = inbound pipe in adapter
        sys_mem:1,              ; - bit 14 - in system unit
                                ;   0 = outbound pipe in system
                                ;   1 = inbound pipe in system
        cfg_b13:1,              ; - bit 13 - reserved
        cfg_b12:1,              ; - bit 12 - reserved
        unit_type:4,            ;Bus Unit Type (bits 11-8)
                                ; - 0000 - system unit
                                ; - 0001 thru 0111 - reserved
                                ; - 1000 - adapter type 1
                                ; - 1001 - adapter type 2
                                ; - 1010 thru 1111 - reserved
                                ;Signalling Conditions
        cfg_b7:1,               ; - bit 7  - reserved
        on_timer:1,             ; - bit 6  - on timer expiration
        on_notfull:1,           ; - bit 5  - on full to non-full
        on_deque:1,             ; - bit 4  - on dequeue element
        on_enque:1,             ; - bit 3  - on enqueue element
        on_empty:1,             ; - bit 2  - on pipe empty
        on_full:1,              ; - bit 1  - on pipe full
        on_notempty:1           ; - bit 0  - on empty to non-empty
CFG_OPT ends

;*  Fields within the Configuration Record (CFG_RCD) are used
;*  at initialization time to identify the location of control areas
```

```
;*  in shared memory address space and the options to be used when
;*  using the delivery service between a pair of bus units.

CFG_RCD struc                   ;Configuration Record
   fid          dw  ?           ;- format identifier
   len          dw  ?           ;- length field
   status       dw  ?           ;- configuration status field
   sid          db  ?           ;- system unit id field
   aid          db  ?           ;- adapter id field
   ss_addr      dd  ?           ;- system signalling address field
   as_addr      dd  ?           ;- adapter signalling address field
   peer_io_addr dw  ?           ;- base I/O address - peer (from)
   base_io_addr dw  ?           ;- base I/O address - adapter (to)
   smgmt_eid    db  ?           ;- system manager entity id field
   time_units   db  ?           ;- units of time
   time_freq    db  ?           ;- timer frequency
   sys_config   CFG_OPT <>      ;- system unit configuration options
   adp_config   CFG_OPT <>      ;- adapter configuration options
   out_size     dw  ?           ;- out-bound pipe size
   in_size      dw  ?           ;- in-bound pipe size
   in_pipe      dd  ?           ;- in-bound pipe address
   in_sds       dd  ?           ;- in-bound surrogate dequeue
                                ;     status area
   in_sse       dd  ?           ;- in-bound surrogate start of
                                ;     elements control area
   in_ses       dd  ?           ;- in-bound surrogate enqueue
                                ;     status area
   in_ssf       dd  ?           ;- in-bound surrogate start of
                                ;     free control area
   out_pipe     dd  ?           ;- out-bound pipe address
   out_sds      dd  ?           ;- out-bound surrogate dequeue
                                ;     status area
   out_sse      dd  ?           ;- out-bound surrogate start of
                                ;     elements control area
   out_ses      dd  ?           ;- out-bound surrogate enqueue
                                ;     status area
   out_ssf      dd  ?           ;- out-bound surrogate start of
                                ;     free control area
CFG_RCD ends

;*
;*  Flags within Local Enqueue Status Flag (LES_FLAGS) are used by
;*  Enqueue Logic to maintain local state information at enqueue
;*  end of the pipe.

LES_FLAGS RECORD                ;Local Enqueue Status Word Flags
        lesb15:1,               ;- bit 15 - reserved
        lesb14:1,               ;- bit 14 - reserved
        lesb13:1,               ;- bit 13 - reserved
        lesd12:1,               ;- bit 12 - reserved
        lesqueued:1,            ;- bit 11 - element stored
        lesb10:1,               ;- bit 10 - reserved
        lesb9:1,                ;- bit 9  - reserved
        leswrap:1,              ;- bit 8  - enqueue wrap
        lesb7:1,                ;- bit 7  - reserved
        lesb6:1,                ;- bit 6  - reserved
        lesb5:1,                ;- bit 5  - reserved
        lesb4:1,                ;- bit 4  - reserved
```

```
        lesb3:1,                ;- bit 3  - reserved
        lesb2:1,                ;- bit 2  - reserved
        lesempty:1,             ;- bit 1  - empty
        lesfull:1               ;- bit 0  - queue full
LES_FLAGS ENDS

;*  Fields within the Local Enqueue Control Area (LECA) identify
;*  the location of the pipe, indicate the status of the pipe, and
;*  provide state information identifying the starting and ending
;*  offsets of control elements within the pipe.

LECA    STRUC                   ;Local Enqueue Control Area
        base    DD  ?           ;- Pipe Address
        we      DW  ?           ;- offset to wrap element
        status  LES_FLAGS <>    ;- enqueue status
        sf      DW  ?           ;- start of free space in pipe
        ef      DW  ?           ;- end of free space in pipe
        end     DW  ?           ;- offset to end (of queue space)
        top     DW  ?           ;- offset to top (wrap point)
LECA    ENDS

;*  Flags within Surrogate Enqueue Status Word (SES_WRD) are used
;*  by Enqueue Logic to convey state of the pipe to Dequeue Logic
;*  at the other end of the pipe.

SES_FLAGS RECORD                ;Surrogate Enqueue Status Word Flags
        sesb15:1,               ;- bit 15 - reserved
        sesb14:1,               ;- bit 14 - reserved
        sesb13:1,               ;- bit 13 - reserved
        sesb12:1,               ;- bit 12 - reserved
        sesb11:1,               ;- bit 11 - reserved
        sesb10:1,               ;- bit 10 - reserved
        sesb9:1,                ;- bit 9  - reserved
        seswrap:1,              ;- bit 8  - wrap of pipe occurred
        sesb7:1,                ;- bit 7  - reserved
        sesb6:1,                ;- bit 6  - reserved
        sesb5:1,                ;- bit 5  - reserved
        sesb4:1,                ;- bit 4  - reserved
        sesb3:1,                ;- bit 3  - reserved
        sesb2:1,                ;- bit 2  - reserved
        sesb1:1,                ;- bit 1  - reserved
        sesfull:1               ;- bit 0  - pipe is full
SES_FLAGS ENDS

;*  Fields within the Surrogate Enqueue Control Area (SECA)
;*  are used by the Enqueue Logic to provide status and state
;*  information to the Dequeue Logic at the other the pipe.

SECA    STRUCT                  ;Surrogate Enqueue Control Area
        status  SES_FLAGS <>    ;- surrogate enqueue status
        ssf     DW  ?           ;- offset to start of free space
SECA    ENDS

;*  Flags within Local Dequeue Status Word (LDS_WRD) are used by
;*  the Dequeue Logic to maintain local state information at the
;*  dequeue end of pipe.

LDS_FLAGS RECORD                ;Local Dequeue Status Flags
```

```
        ldsb15:1,               ;- bit 15 - reserved
        ldsb14:1,               ;- bit 14 - reserved
        ldsb13:1,               ;- bit 13 - reserved
        ldsb12:1,               ;- bit 12 - reserved
        ldsdequeued:1,          ;- bit 11 - element removed
        ldsprempt:1,            ;- bit 10 - stop dequeue operation
        ldsb9:1,                ;- bit 9  - reserved
        ldswrap:1,              ;- bit 8  - wrap element toggle
        ldsb7:1,                ;- bit 7  - reserved
        ldsb6:1,                ;- bit 6  - reserved
        ldsb5:1,                ;- bit 5  - reserved
        ldsb4:1,                ;- bit 4  - reserved
        ldsb3:1,                ;- bit 3  - reserved
        ldsb2:1,                ;- bit 2  - reserved
        ldsempty:1,             ;- bit 1  - pipe is empty
        ldsfull:1               ;- bit 0  - full
LDS_FLAGS ENDS

;*  Fields within Local Dequeue Control Area (LDCA) identify
;*  the location of the pipe, indicate the status of the pipe, and
;*  provide state information identifying the starting and ending
;*  offsets of control elements within the pipe.

LDCA    STRUC                   ;Local Dequeue Control Area
        base    DD  ?           ;- Pipe Address
        we      DW  ?           ;- offset to wrap element
        ds      LDS_FLAGS <>    ;- Dequeue status
        se      DW  ?           ;- start of elements in pipe
        ee      DW  ?           ;- end of elements in pipe
        end     DW  ?           ;- offset to end (of queue space)
        top     DW  ?           ;- offset to top (wrap point)
LDCA    ends

;*  Flags within Surrogate Dequeue Status Word (SDS_WRD) are used
;*  by the Dequeue Logic to convey the state of the pipe to the
;*  Enqueue Logic at the other end of the pipe.

SDS_FLAGS RECORD                ;Surrogate Dequeue Status Flags
        sdsb15:1,               ;- bit 15 - reserved
        sdsb14:1,               ;- bit 14 - reserved
        sdsb13:1,               ;- bit 13 - reserved
        sdsb12:1,               ;- bit 12 - reserved
        sdsb11:1,               ;- bit 11 - reserved
        sdsb10:1,               ;- bit 10 - reserved
        sdsb9:1,                ;- bit 9  - reserved
        sdswrap:1,              ;- bit 8  - wrap of pipe occurred
        sdsb7:1,                ;- bit 7  - reserved
        sdsb6:1,                ;- bit 6  - reserved
        sdsb5:1,                ;- bit 5  - reserved
        sdsb4:1,                ;- bit 4  - reserved
        sdsb3:1,                ;- bit 3  - reserved
        sdsb2:1,                ;- bit 2  - reserved
        sdsempty:1,             ;- bit 1  - pipe is empty
        sdsb0:1                 ;- bit 0  - reserved
SDS_FLAGS ENDS

SDCA    STRUC                   ;Surrogate Dequeue Control Area
        sds     SDS_FLAGS <>    ;- surrogate dequeue status
```

```
        sse     DW  ?           ;- offset to start of elements
SDCA    ENDS

;*  Fields within the Signalling Area (SIGNAL) are used to identify
;*  the source as well as the reason for signalling from one bus unit
;*  to another.

SCA     STRUC                   ;Signalling Control Area
        nque    DB  ?           ;- enqueue state change
        mgmt    DB  ?           ;- management state change
        byte2   DB  ?           ;- reserved
        dque    DB  ?           ;- dequeue state change
SCA     ENDS
```

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                                                                       *
;* INCLUDE FILE: ELEMENT.INC                                             *
;*                                                                       *
;*    COPYRIGHT: (C) Copyright IBM Corporation 1989, 1990.  All          *
;*               rights reserved.                                        *
;*                                                                       *
;*    REV LEVEL: 1.1                                                     *
;*                                                                       *
;*  Description: Contains the data structure definitions for the         *
;*               Move Mode control elements.                             *
;*                                                                       *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

;*        ----- Format Identifier field values -----

FID_MM  equ     0               ;SCB Move mode format

;*        ----- Control Element Type field Element IDs -----

REQ_EL  equ     0               ;request
REP_EL  equ     1               ;reply
EVT_EL  equ     2               ;event
ERR_EL  equ     3               ;error

ERR_EL_LEN equ  20              ;length of error element
DEL_EL_LEN equ  6               ;length of delivery wrap element

;*       ----- Control Element Type field Function Codes -----

                                ;0 and 1 are reserved
FC_INIT    equ  2               ;initialize
                                ;3 is reserved
FC_READ    equ  4               ;read
FC_READL   equ  5               ;read list
FC_READI   equ  6               ;read immediate
FC_WRITE   equ  7               ;write
FC_WRITEL  equ  8               ;write list
FC_WRITEI  equ  9               ;write immediate
FC_EXL     equ 10               ;execute list
FC_MARK    equ 11               ;mark
FC_CANCEL  equ 12               ;cancel
FC_PARAM   equ 13               ;parameters
FC_RDCFG   equ 14               ;read configuration
FC_DIAG    equ 15               ;run diagnostics
                                ;16 is reserved
FC_RESUME  equ 17               ;resume (event)
FC_NOTIFY  equ 18               ;notification (event)
FC_INFORM  equ 19               ;user status information (event)
                                ;20 thru 62 are reserved
FC_WRAP    equ 63               ;wrap of queue (internal)

;       ----- Control Element Value field Error Codes -----

NO_RECV_PENDING equ 1


/*
```

```
                +--------------------------------------+
                |           DATA STRUCTURES            |
                +--------------------------------------+
*/
CIND_FLD RECORD                 ;common indicators field
        id:2,                   ;element id (bits 15-14)
                                ;- (00) request
                                ;- (01) reply
                                ;- (10) event
                                ;- (11) error
        suppress:1,             ;suppress reply flag (bit 13)
        chain:2,                ;chaining flags (bits 12-11)
                                ;- (00) no chaining
                                ;- (01) start of chain
                                ;- (11) middle of chain
                                ;- (10) end of chain
        indirect:1,             ;- bit 10 indirect flag
        notify:1,               ;- bit 9 notification flag
        wait:1,                 ;- bit 8 wait flag
        expedite:1,             ;- bit 7 expedited flag
        fc:7                    ;function codes (bits 6-0)
CIND_FLD ENDS

ADDR_FLD STRUC                  ;Source/Destination Field
        eid     DB  ?           ;- entity identifier
        uid     DB  ?           ;- unit identifier
ADDR_FLD ENDS

ELEMENT STRUC                   ;Element descriptor
        fid     DW  ?           ;- format identifier
        len     DW  ?           ;- length field
        type    CIND_FLD <>     ;- common indicators field
        res1    DW  ?           ;- reserved for future use
        dest    ADDR_FLD <>     ;- destination field
        srce    ADDR_FLD <>     ;- source field
        cor_id  DD  ?           ;- correlation field
        val_fld DD  ?           ;- value field
ELEMENT ENDS
```

Index

D

E

F

H

I

L

M

N

O

P

Q

R

S

T

U

V

W