INTERCOMM

DYNAMIC DATA QUEUING FACILITY

ISOGON

330 Seventh Avenue · New York, New York 10001

Dynamic Data Queuing Facility

Publishing History

Publication	Date	Remarks
First Edition	June 1974	This manual corresponds to Intercomm Release 6.0.
Second Edition	January 1975	General updates, corresponding to Intercomm Release 6.1.
Third Edition	February 1979	General updates, additions and format changes, corresponding to Intercomm Release 8.0.
Fourth Edition	July 1999	Updates and revisions corresponding to Intercomm Releases 9, 10 and 11.

Notes: the following are not supported under Release 9:

DDQPRT Utility
VS COBOL II
Subsystem loading above the 16M line.
Backout-on-the-fly feature of the File Recovery Facility in more than one region in a Multiregion environment

PREFACE

Intercomm is a state-of-the-art teleprocessing monitor system executing on the IBM System/390 family of computers and operating under the control of IBM Operating Systems (ESA and OS/390). Intercomm monitors the transmission of messages to and from terminals, concurrent message processing, centralized access to I/O files, and the routine utility operations of editing input messages and formatting output messages, as required.

The Dynamic Data Queuing Facility (DDQ) is a Special Feature of the Intercomm system. This manual documents the utilization of the DDQ Facility from the point of view of the application program, and details implementation techniques for the System Manager. Since this is a reference document, only DDQ specifications are described.

In conjunction with the use of this document, the reader is referred to the following Intercomm publications:

- Concepts and Facilities
- Operating Reference Manual
- COBOL Programmers Guide
- PL/1 Programmers Guide
- Assembler Language Programmers Guide
- Messages and Codes

The DDQ Facility is also used by the Multiregion special feature (see Multiregion Support Facility), the serial restart facility (see Operating Reference Manual), and optionally for the Message Mapping Utilities (MMU) output message data storage processing (see Message Mapping Utilities).

Note: in this manual, the term COBOL refers to all supported COBOL compilers, a distinction is made only if necessary.

INTERCOMM PUBLICATIONS

GENERAL INFORMATION MANUALS

Concepts and Facilities

Planning Guide

APPLICATION PROGRAMMERS MANUALS

Assembler Language Programmers Guide

COBOL Programmers Guide

PL/1 Programmers Guide

SYSTEM PROGRAMMERS MANUALS

Basic System Macros

BTAM Terminal Support Guide

Installation Guide

Messages and Codes

Operating Reference Manual

System Control Commands

CUSTOMER INFORMATION MANUALS

Customer Education Course Catalog

Technical Information Bulletins

User Contributed Program Description

FEATURE IMPLEMENTATION MANUALS

Autogen Facility

ASMF Users Guide

DBMS Users Guide

Data Entry Installation Guide

Data Entry Terminal Operators Guide

Dynamic Data Queuing Facility

Dynamic File Allocation

Extended Security System

File Recovery Users Guide

Generalized Front End Facility

Message Mapping Utilities

Multiregion Support Facility

Page Facility

Store/Fetch Facility

SNA Terminal Support Guide

Table Facility

TCAM Support Users Guide

Utilities Users Guide

EXTERNAL FEATURES MANUALS

SNA LU6.2 Support Guide

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Chapter 1

INTRODUCTION

1.1 GENERAL DESCRIPTION

The Dynamic Data Queuing Facility (DDQ) provides application programs with the ability to dynamically create, retrieve, and delete logical data sets and/or queues of messages on a single BDAM data set. This eliminates the need to define separate data sets for small, transient queues of data and/or messages. As used in this manual, the terms "queue" and "dynamic data queue" refer to any logical sequence of records, irrespective of size or content.

1.2 DDQ FEATURES

The DDQ Facility provides the following features:

- Dynamic queues of practically any size may be created for use on-line with Intercomm or in batch mode. The size of DDQ records written on dynamic queues may be up to 32K minus 5 bytes (DDQ processing limition).
- Either message or non-message data may be contained in queue records.
- Queues may be created, updated, added to, read and deleted via the DDQ Facility service routines, requested via application program logic.
- Queues may be saved for any period of time, and retrieved at any time via user-specified queue identifiers.
- Queues may be created on one or more BDAM data sets. If multiple BDAM data sets are available, the application program may force the creation of a queue on a particular data set, or use a default data set.
- Records on queues may be blocked or unblocked. Automatic blocking/deblocking of blocked records is provided.
- Automatic handling of variable-length records is provided, including handling of records larger than the physical block size of the data set. This is true for both blocked and unblocked queues.

- Queues may be shared between on-line and batch programs.
- The space allocated to a queue for a subsystem message processing thread may be recovered, via the Resource Management Audit and Purge facility, after subsystem failure.
- Queues may be preserved if Intercomm is restarted using the standard restart facility.
- Queues may be specified as single-retrieval or multiple retrieval. Multiple retrieval queues may be read non-destructively as many times as desired.
- A DDQ containing messages prepared for transmission can be passed directly to the Front End using the Front End Control Message (FECM) facility.

1.3 USES OF DYNAMIC DATA QUEUING

1.3.1 Segmented Messages

Full multithreading of Intercomm segmented input and output messages to and from application subsystems can be effected. Since each dynamic queue is unique to a message thread, no interleaving of unrelated message segments will occur. This obviates the need to lock out a terminal until a subsystem has read or written the final segment of a multi-segment message.

1.3.2 Data Collection

Application subsystems have the option of accumulating data for a period of time before processing that data, either on-line or in batch mode. They may do this by building a dynamic queue and adding data records to the queue when necessary.

1.3.3 Message Collection

Application subsystems have the option of delaying processing/outputting of messages until a time period has elapsed or a certain event has occurred. The messages are collected on a DDQ until they are to be processed, or transmitted.

1.3.4 Data Switching

If large amounts of data are to be passed from one subsystem to another, the normal Intercomm message switching scheme may be unwieldy. The DDQ Facility enables the passing of data by building a queue of records containing the data, and then forwarding the queue to the destination subsystem.

1.3.5 Inter-Subsystem Communications Area

If many subsystems access a large common area of data, the area may be written out as a dynamic queue, and retrieved for updating via a common Queue Identifier (QID).

1.3.6 Temporary Storage

Subsystems may at times store large data areas, in order to free some main storage, if their processing paths do not require the data for a considerable period of time.

1.3.7 Minimizing Storage Requirements

24-Amode Assembler Language subsystems may minimize their main storage requirements during calls to CONVERSE by writing out most of their dynamic work areas to a dynamic queue.

1.3.8 Batch/On-Line Communications

Queues created by an on-line program may be accessed via a batch program and vice versa.

1.4 FUNCTIONAL CHARACTERISTICS

The significant characteristics and user options of the DDQ Facility are tabulated in the following subsections.

1.4.1 Queue Types

When a new dynamic queue is created, the desired type of queue must be specified: transient, single-retrieval transient, semi-permanent or permanent:

- Transient queues

  These are queues of data/messages that are not to be saved for later retrieval. Once created, a transient queue must be freed or passed to another subsystem. Transient queues are not preserved across an Intercomm restart.

- Single-Retrieval Transient Queues

  These are queues that are identical to transient queues but may be retrieved one time only. The advantage of this type of queue is that the space for these queues is freed as soon as possible after retrieval, rather than at queue close time.

- Semi-permanent Queues

  These are queues that the user wishes to retrieve, via a user-provided queue identifier, at a later point in time. Some additional I/O overhead is involved in saving the queue. This type of queue can be freed by explicit user request. Semi-permanent queues are optionally preserved across an Intercomm restart. At normal startup, existing semi-permanent queues are freed.

- Permanent Queues

  These are identical to semi-permanent queues, except permanent queues are always preserved across any Intercomm startup, warm or cold. Permanent queues must be freed by explicit user request.

For a queue to be preserved and available at restart, it must be completed by the creating program, that is, the queue must have been closed.

1.4.2 Queue Records

Records written to a dynamic data queue should consist of data prefixed with a four-byte length field. The length must be in the first two bytes of the prefix. Each record may be of any length up to 32K (less 5 bytes). If the data record is a message with a preformatted header containing the length in the first 2 bytes, then the 4-byte length field is not needed (may not be used).

1.4.3 Queue Access

Normal queue access is sequential. This may be varied in the following ways:

- Updating

  The application program may request that a record be read and written back. All updates take effect immediately. The length of the record cannot be changed.

- Adding Records to the Front of a Queue

  The application program may request that a particular record be placed in logical sequence ahead of existing queue records.

- Retrieval by Relative Record Number

  The application program may retrieve queue records at random by supplying a number corresponding to the record's position in the queue; that is, to retrieve the first record on a queue, the relative record number is 1, for the nth record, the relative record number would be n. This type of retrieval can only be used with unblocked queues, and the largest record on a queue must not exceed the block size of the queuing data set. If sequential retrieval is performed prior to a read by the relative record number, specific retrieval does not alter the read position on the queue for subsequent sequential retrieval.

- Retrieval by Record Location

  When writing a record to a queue, feedback information on the record's location in the queue may be requested with either blocked or unblocked queues. This information is used to retrieve the record at a later time. When a record is retrieved in this manner, the reading position in the queue is then set for a subsequent read of the record following the one requested. If the next read request does not specify a relative record number or a record location field, the record read is the one following the last record read by location. This function is similar to a QISAM SETL function. The record location returned by the DDQ facility is in internal DDQ format and must not be changed in any way.

1.4.4 Queue Size

The maximum number of records that can be written to a queue is a function of the available blocks on the queuing data set and of the queue size. When a queue is first created, the application program specifies the number of blocks to be allocated to the queue (or uses the default allocation value for that data set). If the queue's space is then exhausted, the Dynamic Data Queuing Facility will automatically allocate another extent of the same size as the initial (or primary), extent. A queue may own up to l6 space extents. The acquisition of extents is contingent upon having available space on the queuing data set.

1.4.5 Queue Location

Dynamic data queues reside on BDAM data sets, preformatted by the CREATEGF utility program. At least one data set must be provided for queue residency. Multiple data sets may be used to build queues, and a queue can be forced to be created on a particular data set, or else use the default data set. In this manual, these data sets are referred to as queuing data sets.

1.4.6 Queue Recovery

Non-transient queues may be recovered if restart/recovery is used. Recovery of queues is independent of the Intercomm File Recovery Special Feature.

1.4.7 Queue Resource Management

If Resource Auditing is in effect, the DDQ logical data set (queue) is monitored as an active resource of a message thread between the thread's initial request to create or access a queue and the termination of access specifying queue disposition by that thread. If a subsystem thread terminates (abnormally or normally) without indicating queue disposition, the associated queue space is freed if a queue of any type was being built (created) or if the type is transient. If an opened queue is semi-permanent or permanent, then the queue is kept (including thread updates), except that any new extents added by the thread will be deleted.

1.4.8 Queue Records Automatically Input to a Subsystem

The DDQ interface module (DDQINTFC) may be utilized to intercept messages directed to a subsystem and check for the presence of a special message indicating a dynamic data queue exists for the subsystem. If the special message is present, DDQINTFC automatically retrieves messages from the queue and passes them to the subsystem for processing. In this mode of operation, no subsystem logic is required to retrieve the messages from the DDQ. If the presence of a DDQ message is not found, normal message processing proceeds. When processing of the DDQ is completed, the DDQ will be freed if transient, otherwise it will be saved.

Queues containing data records (not messages) may not be retrieved by DDQINTFC.

This facility is not supported for subsystems loaded above the 16M line, nor for any COBOL subsystems.

See Chapter 3 for detailed specifications on using DDQINTFC.

1.4.9 Front End Data Queuing

Through the use of a Front End Control Message (FECM), Front End Data Queuing enables an application to pass a DDQ directly to the Front End. The DDQ contains messages ready to be transmitted to the terminal specified in the message header of the FECM. Groups of related output messages may be put on a single DDQ. Therefore, requesting of Front End terminal-oriented queuing of the individual messages is avoided. Also, the possibility of overflowing the terminal queue (if the terminal is down or disconnected) is minimized.

In order to use this feature, a permanent or semi-permanent DDQ is built containing messages that are preformatted with the necessary control characters for the destination terminal. The DDQ may also contain FECMs for other DDQs, or feedback FECMs, with the output messages. The various other types of FECM messages and how to create them is described in the Programmers Guides.

After the queue is built, a FECM is formatted by calling the entry point FECMDDQ of the FECMMOD module. (See Chapter 2 for detailed specifications.) The DDQ is passed to the Front End by queuing the FECM formatted by FECMDDQ for the Output subsystem as a VMI X'67' message, or by passing the FECM to the Front End via the FESEND (FESENDC) service routine. If Message Mapping (MMU) is used by an application subsystem to format a group of output messages, it will automatically create the DDQ and queue a FECM for the Front End, if requested.

Nesting of DDQs is possible if the FECM obtained above is written to another DDQ, which is in turn passed to the Front End. A broadcast group terminal-id may be used in the FECM message header if all terminals in the group are the same type (that is, all 3270s, etc.).

1.4.10 Other Features

Recommendations for defining a DDQ data set for use with the Message Mapping Utilities (MMU) are given at the end of Chapter 4.

Gathering of statistics on DDQ CALL activity and data set I/O is described in Chapter 4.

Printing of the contents of semi-permanent and permanent DDQs via the DDQ Print Utility (DDQPRT) is described in Chapter 5. This feature is useful for checking the contents of subsystem-created DDQs or to determine why a DDQ data set is filling up.

The linkedit for a batch program which accesses, creates, or deletes DDQs is listed in Chapter 4. Creation of DDQ data sets and execution JCL is also detailed in Chapter 4, and Multiregion environment considerations are also given.

Chapter 2

APPLICATION PROGRAM INTERFACE

2.1 INTRODUCTION

This chapter describes application program interface with the Dynamic Data Queuing Facility. It documents the following subjects:

- Using the DDQ Facility with Intercomm
- Linkage
- Creating a new queue (QBUILD)
- Selecting an existing queue (QOPEN)
- Queue disposition (QCLOSE)
- Writing to a queue (QWRITE)
- Reading from a queue (QREAD)
- Updating a queue record (QREADX/QWRITEX)
- Creating a Front End Control Message (FECMDDQ)
- Subsystem design restrictions

2.2 USING THE DDQ FACILITY WITH INTERCOMM

2.2.1 Using the DDQ Facility via the Standard CALL Statement

An Intercomm subsystem or a batch program uses the DDQ Facility via the standard CALL statement. The following functions may be called to perform the corresponding operation:

- QBUILD--create a queue and define its attributes
- QOPEN--prepare DDQ control blocks for application program access of an existing queue

- QCLOSE--indicate application program access of a queue is complete and define its disposition (free, pass or save)
- QWRITE--add a record to a queue
- QREAD--retrieve a record from a queue
- QREADX--retrieve a record from a queue with intent to update
- QWRITEX--update a record on a queue

These functions are entry points in the DDQ monitor module, DDQMOD.

The entry point FECMDDQ of the FECMMOD module is called to produce a Front End Control Message (FECM) for Front End Data Queuing.

2.2.2 Building a DDQ

When a DDQ is built, a 48-byte Queue Locate Block (QLB) is constructed containing the Queue Identifier (QID) and other internal control information. This QLB is referenced in all subsequent calls to DDQ.

2.2.3 Closing a DDQ

When a DDQ is closed, it is freed, saved or passed to another subsystem, based upon parameter specifications passed to the QCLOSE routine. Non-transient queues are saved if they are not freed, and control information is written to the DDQ Queue Control File (QCF) for potential use at system restart time and/or for delayed or batch access to the queue.

2.2.4 Passing a DDQ to Another Subsystem

If a DDQ is to be passed to another subsystem when QCLOSE is called, DDQ generates a message for the receiving subsystem consisting of a standard message header with VMI X'EE' and message text consisting of the 48-byte Queue Locate Block (QLB). The receiving subsystem may then process the DDQ in one of two ways. The subsystem checks each input message for the VMI X'EE' message, and then uses DDQ queue processing logic to retrieve the queued data; or, a non-COBOL subsystem may use the Automatic Queue Input facility (DDQINTFC) to process each entry on the queue as an independent message.

The first case is illustrated in Figure 1.

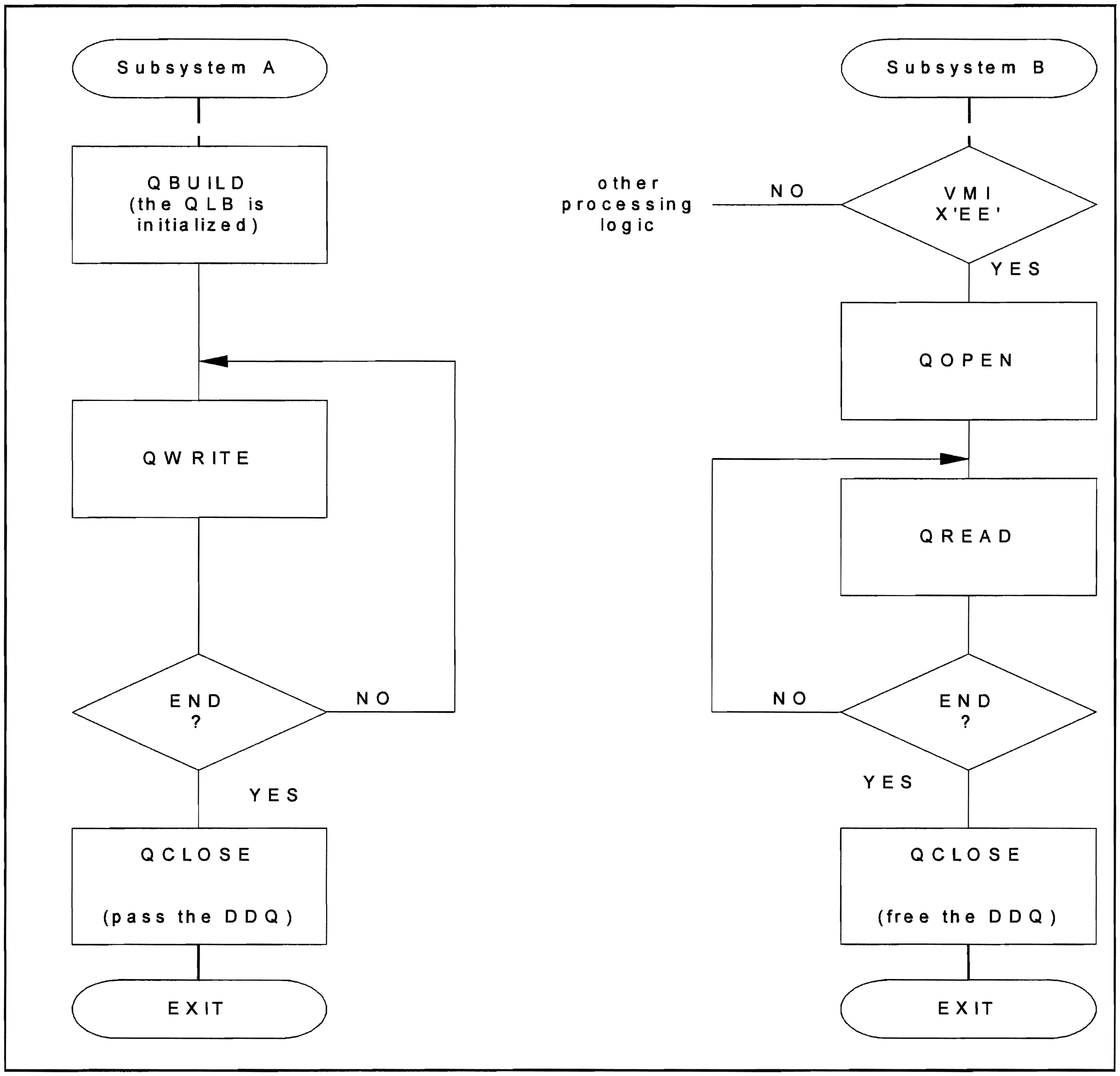


Figure 1. Subsystem Logic for Passing a DDQ

This technique is appropriate when the queue data consists of related pieces of information, such as in an order entry situation where the queue data represents related items of one order. Subsystem A gathers onto a queue the order information entered by the terminal operator; Subsystem B, in turn, processes the entire order (queue) for inventory control purposes, shipping orders, etc.

The second case, in which the subsystem uses the Automatic Queue Input facility, is illustrated in Figure 2.

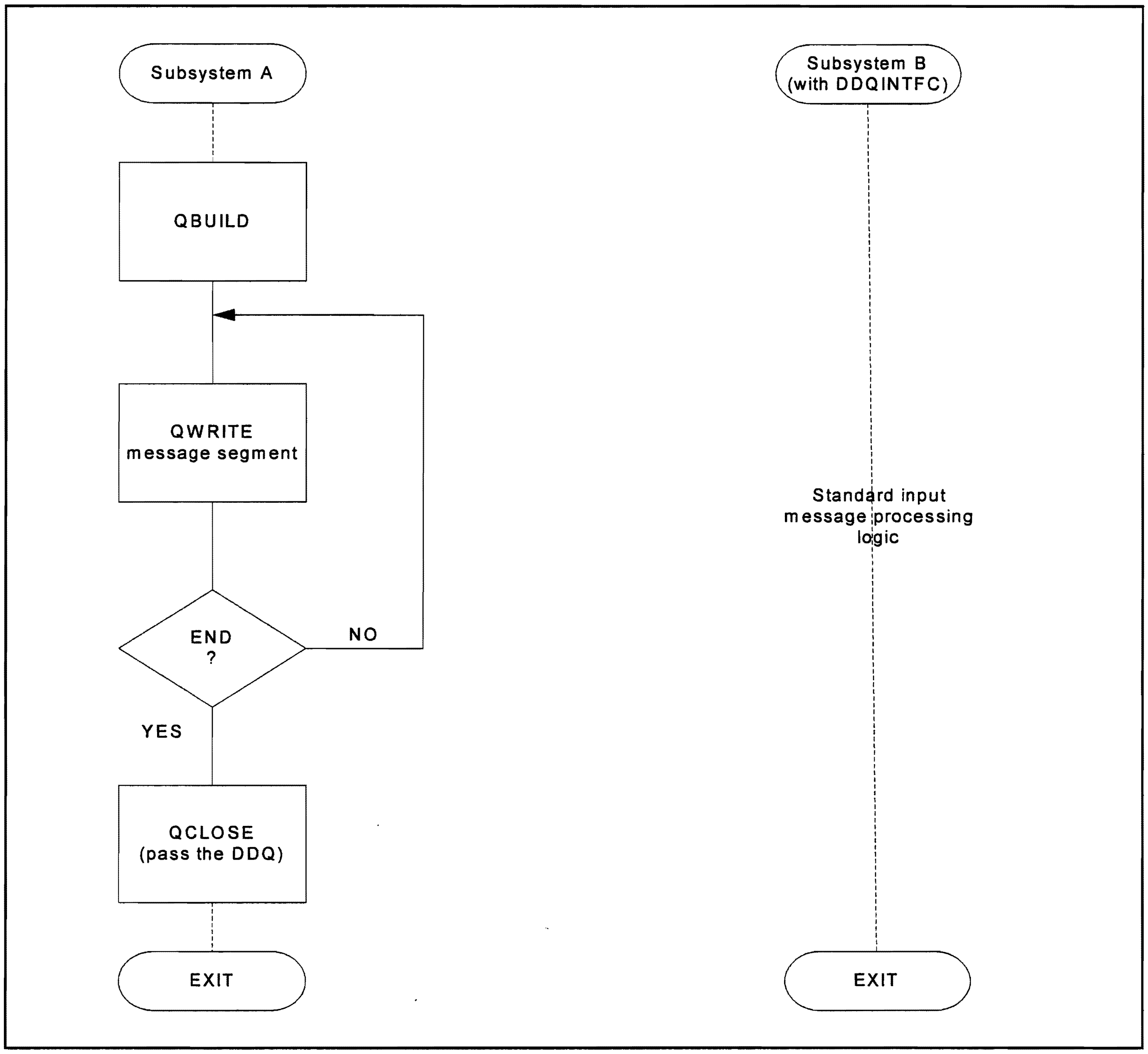


Figure 2. Subsystem Logic for Passing a DDQ and Using the Automatic Queue Input Facility

Here the DDQ is used for segmented output messages. Application subsystems create segmented messages for processing by another subsystem (or Output Utility) and route those messages to a dynamic data queue, rather than the normal Intercomm subsystem queue. The receiving subsystem then multithreads each queue of segmented messages using standard Intercomm message processing logic. The messages within a queue are processed serially. Without a DDQ, multithreaded retrieval of segmented messages from the same subsystem queue occurs. (If the Output Utility uses the DDQ Automatic Queue Input facility, it need not be modified to retrieve messages from dynamic queues. See Chapter 3.)

2.2.5 Retrieval of Data from a Saved Queue

To make possible retrieval of data from a saved queue, the application program must communicate the Queue Identifier (QID) to terminal operators. For example, in an order entry system, a DDQ can be used to hold orders entered by an operator at the terminal. The DDQ can be defined as a permanent queue to allow access to the order information until the data is no longer needed.

The DDQ can be freed based upon program logic and/or an input transaction. When the order is entered, the operator at the originating terminal is notified of the QID. Subsequent messages may be entered from the terminal requesting retrieval or update of order data. Figure 3 illustrates three types of transactions in this environment.

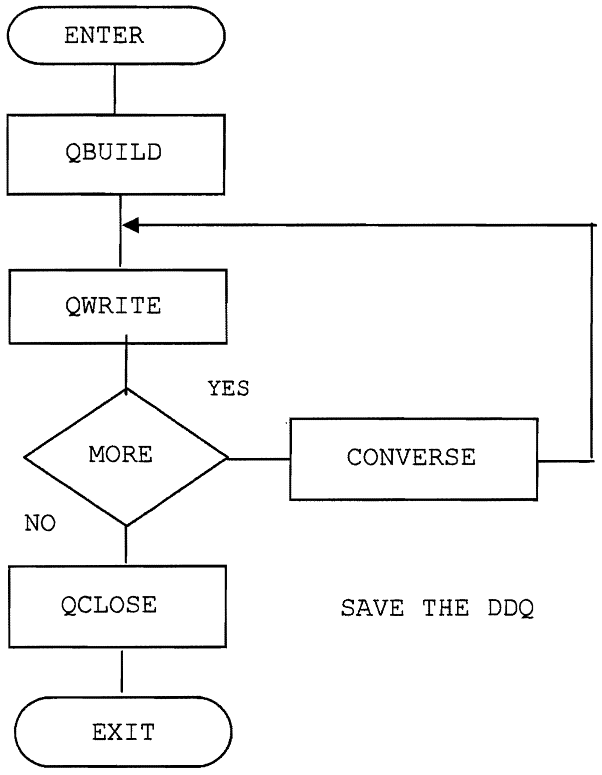
2.2.6 Front End Data Queuing

Front End Data Queuing is accomplished through the Front End Control Message (FECM) facility. An application subsystem builds a DDQ consisting of preformatted messages (VMI=X'57', if preformatted by the subsystem logic) or mapped messages (VMI=X'67', after processing by Message Mapping Utilities). The entire DDQ is passed to the Front End for transmission by creating a FECM and passing it to the Front End. The FECM is then queued as though for transmission to the terminal. When the Front End retrieves the FECM from the terminal-oriented queue, the FECM is not sent to the terminal, but triggers the processing of the DDQ. The messages in the queue are then serially transmitted to that terminal. An example of application logic is illustrated in Figure 4.

Alternatively, the subsystem using MMU may request at MAPEND time that all mapped messages be put on a DDQ and a FECM automatically be created and queued for the terminal. Under Releases 10 and 11, multiple copies of the entire series of messages in a DDQ may be transmitted to the same terminal (for example, a report to a printer) via an additional MAPEND ‘copies’ parameter (see Message Mapping Utilities).

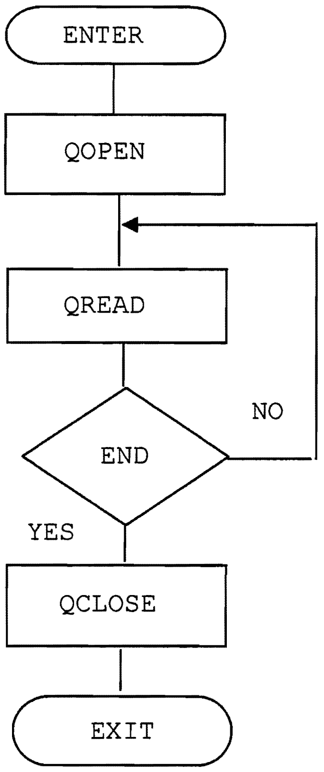
ORDER ENTRY

```
Input message(s): order data
Output message:   order control
                  number (QID)
```



STATUS REQUEST

```
Input message(s): QID
Output message:   order control
                  number (OID)
```



SHIPPING TRANSACTIONS

```
Input message(s): QID and shipping data
Output message:   verification of update
```

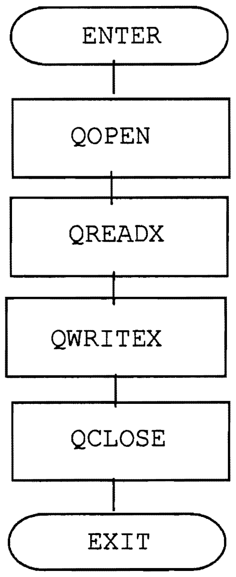
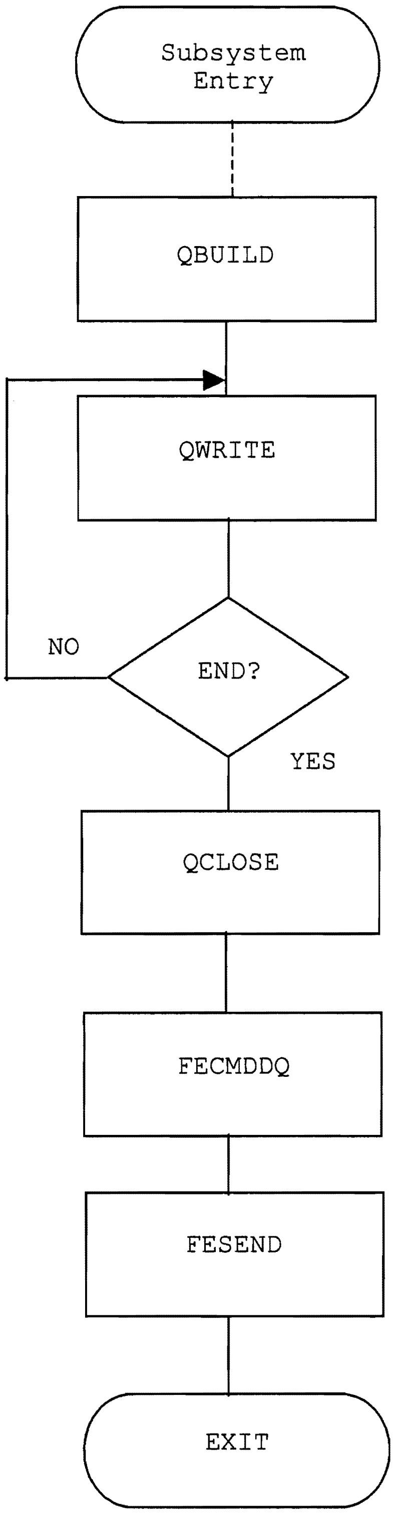


Figure 3. Three Types of Transactions in the DDQ Environment



FECMDDQ creates the FECM with all required control data in the message text.

FESEND is used, instead of queuing the FECM for processing by the Output Utility. It passes the message to the Front End to be queued on a terminal queue.

Figure 4. Application Logic for Front End Data Queuing (FECM)

2.3 LINKAGE

2.3.1 User-Supplied Parameters

When calling a DDQ function, the user must supply a set of parameters. The parameter areas must be in a reentrant program's dynamic work space (except if the caller is VS COBOL II). The following parameters are passed:

- qlb

  qlb (a required parameter) is the address of the Queue Locate Block (QLB). This is a 12-fullword (48 byte) area, on a fullword boundary, used by the DDQ Facility to hold information about the queue being processed. The first l6 bytes of this area contain the QID, and may be initialized for QBUILD and QOPEN requests when the user is supplying the QID; otherwise, the QLB must never be modified by a user program.

- qsw

  qsw (required) is the address of the Queue Status Word (QSW), a fullword (four-byte) area, on a fullword boundary. The QSW is used to modify a function call and to receive return codes from the DDQ Facility. All return codes placed in the QSW can also be found in Register 15, in binary and multiplied by 4. Assembler Language programs can code a branch table following any call to a DDQ function.

- qmm

  qmm (required for all functions except QOPEN and QCLOSE) is for READ/WRITE calls; this is the address of an I/O area. Otherwise, it is the address of a function-related parameter.

- qnn

  qnn (optional) is the address of a function-related parameter.

2.3.2 Calling DDQ Functions

For COBOL and optionally for PL/1, the service routines are invoked through calls via the interface routines COBREENT or PMIPL1 (not required for PL/1 – function may be called directly). The code names for the functions are provided in the COPY members ICOMSBS (COBOL) and PENTRY or PLIENTRY for PL/1. Note that in ICOMSBS, each function name has a prefix, for example DQ-BUILD for QBUILD, etc. The REENTSBS codes (and relative offsets) for DDQ functions are as follows:

- QBUILD--3
- QOPEN--7
- QCLOSE--11
- QREAD--15
- QWRITE--19
- QREADX--23
- QWRITEX--27

The specific form of the CALL statement depends on the programming language being used. Examples are given below for all languages supported by Intercomm ('function' is the specific DDQ routine being called):

- Nonreentrant Assembler Language:

```
CALL  function,(qlb,qsw[,qmm][,qnn]),VL
```

- Reentrant Assembler Language:

```
CALL  function,(qlb,qsw[,qmm][,qnn]),VL,MF=(E,list)
```

- Nonreentrant COBOL:

```
CALL  'function' USING qlb,qsw[,qmm][,qnn].
```

- Reentrant COBOL:

```
CALL  'COBREENT' USING DQ-function,qlb,qsw[,qmm][,qnn].
```

  where DQ-function is the label of the desired REENTSBS offset code

- PL/1:

```
CALL  function(qlb,qsw[,qmm][,qnn]);

                or

CALL  PMIPL1(ddq-function,qlb,qsw[,qmm][,qnn]);
```

  where ddq-function is the label of the desired REENTSBS offset code.

2.4 CREATING A NEW QUEUE (QBUILD)

QBUILD is invoked to create a queue. At this time, the application program defines queue attributes such as size, location, queue type, queue identifier, and blocked/unblocked attribute. The parameter list for QBUILD is:

qlb,qsw,qmm[,qnn]

where:

- qlb (required) is the address of a QLB, which is a 48-byte area on a fullword boundary. The application program may initialize the first 16 bytes of the QLB with a 16-byte QID. If the application program does not supply the QID, the Dynamic Data Queuing Facility assigns a unique QID based on date and time of day.

- qsw (required) is the address of a QSW, a four-byte field on a fullword boundary. This field must be initialized to blanks and/or appropriate function codes prior to calling QBUILD. The function of each byte is as follows:

  -- byte 1: return code from QBUILD

  C'0'--normal completion

  C'1'--duplicate QID--non-transient queue

  C'2'--no space found on the Queue Control File for building queue

  C'3'--invalid ddname, or data set unusable

  C'4'--invalid or conflicting QBUILD options; check coding of DDQDS macro for specified data set, also DDQENV option coding.

  -- byte 2: defines the queue type. Must be initialized by the application program with one of the following codes:

  C'T'--transient queue

  C'S'—semi-permanent queue

  C'P'--permanent queue

  C'V'--single-retrieval transient queue

  -- byte 3: if initialized with a C'B', the queue is blocked (if blocked queues are permitted on the DDQ data set; see DDQDS macro). The default (blank) is an unblocked queue.

- -- byte 4: if initialized with a C'Q', the DDQ Facility uses the first 16 bytes of the QLB as the QID; the application program is responsible for initializing the QLB with the appropriate QID. The default (blank) is that the DDQ Facility assigns a QID and places it in the first 16 bytes of the QLB.

When the application program regains control, after calling QBUILD, bytes 3 and 4 of the QSW will contain a binary value equal to the physical block size of the queuing data set.

- qmm (required) is the address of a four-byte area on a fullword boundary. The first two bytes must be initialized prior to calling QBUILD, as follows:

  -- binary zeros: the number of blocks in a queue extent is to be the default (n) for the data set; see BLOCKS parameter of DDQDS macro.

  -- binary integer: the number supplied (n) becomes the number of physical blocks allocated to each queue extent.

  The value n, whether user-specified or default, is the number of contiguous physical blocks that the DDQ Facility reserves on the DDQ data set for the queue to be built. If a queue extent becomes full, the DDQ Facility allocates up to 15 more extents, each equal to n physical blocks; this allows for a maximum queue size of 16*n blocks (depending on value coded for the EXTNUM global in the DDQENV table (see Appendix B), the default is 16). Secondary allocation should be avoided by supplying a good estimate of the size of the queue. However, over-allocation should also be avoided, as this wastes space on the DDQ data set.

  Bytes 3 and 4 are reserved for future use, and must be binary zeros or blank.

- qnn (optional) is the address of an eight-byte field containing the ddname of the DDQ data set on which the queue is to be built. If this parameter is omitted, the DDQ Facility will build the queue on the default DDQ data set. If used, the ddname field must be in the caller's dynamic working storage area (DWS/DSA or save/work area) if the calling program may be dynamically loaded above the 16M line, except if the caller is VS COBOL II.

2.5 SELECTING AN EXISTING QUEUE (QOPEN)

Before accessing an existing queue, the application program must issue a QOPEN to initialize the QLB and to give exclusive access of the queue to the calling program. This exclusive access must later be relinquished by invoking QCLOSE.

Queues to be opened may be queues passed to a subsystem by either Intercomm or another subsystem, or queues that were previously saved. In the former case, the VMI field of the message header contains a X'EE'. The application program must not change the contents of this message before issuing a QOPEN. A saved queue, on the other hand, may be opened whenever the subsystem logic requires it.

If the application program has just created the queue via QBUILD (within the same processing thread), a QOPEN is unnecessary and should not be issued.

The parameter list for QOPEN is:

qlb,qsw

where:

- qlb (required) is the address of the QLB, a 48-byte area on a fullword boundary. To access a queue, the first 16 bytes of the QLB must be initialized with the QID of that queue, unless the queue is being passed via a VMI X'EE' message. In that case, do not initialize the QLB; the DDQ Facility will locate the QID from the message text itself and perform the QLB initialization.

- qsw (required) is the address of the QSW, a four-byte field on a fullword boundary. The function of each byte is as follows:

  -- byte 1: return code from QOPEN

  C'0'--normal completion

  C'1'--queue not found

  C'2'--queue is already open (in use) by another thread

  C'3'--invalid ddname or unusable data set

  C'4'--message with VMI X'EE' not found

  C'5'--QOPEN was issued for a permanent or semi-permanent queue, but subsystem access to the DDQ data set has been restricted to transient queues by the installation (see DDQDS macro, PERMS parameter)

  -- byte 2: must be initialized with a C'M' when the application program opens a queue with a QID passed via the VMI X'EE' message. Otherwise, must be zero or blank.

  -- bytes 3 and 4: unused--must be binary zero or blank.

  When the application program regains control after calling QOPEN, bytes 3 and 4 of the QSW will contain a binary value equal to the block size of the data set or the length of the largest record on the queue, whichever is larger.

2.6 QUEUE DISPOSITION (QCLOSE)

Dynamic data queues must eventually be disposed of. Queue disposition is accomplished by invoking QCLOSE. There are three ways to dispose of a queue: it can be freed, it can be saved, or it can be passed to another Intercomm subsystem, which may then QOPEN and QCLOSE it.

Queue disposition is also determined by whether or not the queue is transient. Transient queues cannot be saved, but can only be passed or freed. A single-retrieval transient queue accessed via a QOPEN (for retrieval) cannot be passed; it can only be freed. If the application program does not specifically request that a transient queue is to be passed, it is freed. All non-transient queues are automatically saved, unless explicitly freed. If a block of records to be added to the queue was being built in core when QCLOSE is called, it is written to the queue data set before QCLOSE processing is started.

The parameter list for QCLOSE is:

qlb,qsw[,qnn]

where:

- qlb (required) is the address of the QLB provided at QOPEN or QBUILD time.
- qsw (required) is the address of the QSW. The function of each byte is as follows:
  - -- byte 1: return code from QCLOSE

    C'0'--normal completion

    C'1'--I/O error on queuing data set

    C'2'--queue full (last block could not be added)

    C'3'--queue not owned. QOPEN/QBUILD not issued.

    C'4'--queue can not be passed. Probably due to an invalid receiving subsystem code. Other possible causes are: no room on the receiving subsystem queue, or no core for that subsystem's disk queue I/O area.

- byte 2: specifies whether the queue is to be passed or freed, if the default is different.

  C'P'--queue to be passed (requires qnn parameter)

  C'F'--queue to be freed

  C'T'--the QCB is to be written out, but the application program thread can continue processing the queue without having to reopen it. On restart, the queue is restored to its state at the time of this close request, if a normal (P or F) close was not subsequently issued. (This is only applicable to non-transient queues.)

  Any other specification in this byte means the queue will be saved if non-transient, freed if transient.

- bytes 3 and 4: unused--must be binary zeros or blank.

• qnn (required if queue is being passed) is the address of a four-byte or twelve-byte area on a fullword boundary. Each byte must be initialized as follows:

- bytes 1 and 2: the code of the subsystem to receive the queue. (The left-most byte is the high-order code.)

  NOTE: The following three items are only necessary if QCLOSE is not being called by a subsystem (twelve-byte area required). They are used to construct the message header of the VMI X'EE' message. If the caller is a subsystem, this information is taken from the subsystem input message header (as saved by the Subsystem Controller), and these entries will be ignored.

- bytes 3 and 4: the code of the subsystem passing the queue. (The left-most byte is the high-order code.)

- bytes 5 through 9: the terminal-ID to be used (MSGHTID).

- bytes 10 through 12: value to be used for the Front End Sequence Number (MSGHBMN), which may be zero; use only bytes 10 and 11 if a 2-byte BMN number is used system-wide.

2.7 WRITING TO A QUEUE (QWRITE)

Unless the application program specifies otherwise, records are written sequentially using QWRITE. Optionally, the application program may place each record at the head of its queue (except for single-retrieval transient queues). When the queue is sequentially retrieved, the most recently written record at the head of the queue will be the first record read. In this way, a reverse sequential queue may be created by writing all records to the head of the queue. During one thread's access to a queue, records may only be written in one direction, unless the queue is closed and reopened for writing in the opposite direction.

The parameter list for QWRITE is:

qlb,qsw,qmm[,qnn]

where:

- qlb (required) is the address of the QLB, provided when the QOPEN or QBUILD call was issued.
- qsw (required) is the address of the QSW. The function of each byte in the QSW is as follows:
  - -- byte 1: return code from QWRITE

    C'0'--normal completion

    C'1'--I/O error on queuing data set. DDQMOD closes the DDQ.

    C'2'--queue is full

    C'3'--queue is not owned. QOPEN/QBUILD not issued, or queue has been closed.

    C'4'--reserved for future use

    C'5'--invalid record length
  - -- byte 2: if the record is to be written to the head of the queue, this field must be initialized with a C'H'. Otherwise, it should be zero or blank.
  - -- bytes 3 and 4: unused--must be binary zero or blank.
- qmm (required) is the address of the record to be written. The record must be on a halfword boundary, and must (if not an Intercomm message with a formatted header) have a four-byte prefix, with the length of the record (including the prefix) in the first two bytes of the prefix. The length must be a binary number between 5 and 32,767.

- qnn (optional), if specified, must be the address of a four-byte field on a fullword boundary. The DDQ Facility returns the record location of the just-written record in this field. This may later be used for direct retrieval of the record, if supplied unchanged in a QREAD call. This field is called the write feedback field. Feedback information is unobtainable when a record is written to the head of a queue. Therefore, when byte 2 of the QSW is an 'H', this parameter must not be coded. This feature may be used with blocked or unblocked queues.

2.8 READING FROM A QUEUE (QREAD)

Records are retrieved using QREAD. Unless otherwise specified, retrieval of queue records is sequential and commences from the beginning of the queue. If the queue is not a blocked queue, the application program can override sequential retrieval by supplying a relative record number. However, if the specific location obtained via a previous QWRITE with feedback is requested, sequential retrieval from that point can subsequently be requested.

The parameter list for QREAD is:

qlb,qsw,qmm[,qnn]

where:

- qlb (required) is the address of the QLB provided at QOPEN or QBUILD time.

- qsw (required) is the address of the QSW. The function of each byte in the QSW is as follows:

  -- byte 1: return code from QREAD

    C'0'--normal completion

    C'1'--invalid parameter or retrieval option, or I/O error on queuing data set (the DDQ facility closes the queue)

    C'2'--queue is empty or end-of-queue, or invalid qnn read start point value

    C'3'--queue is not owned. QOPEN/QBUILD not issued, or queue has been closed.

  -- byte 2: C'L'--retrieve by specific record location provided by a QWRITE feedback field, otherwise blank or binary zero (retrieve sequentially) - see also qnn below

  -- bytes 3 and 4: unused--must be binary zero or blank.

- qmm (required) is the address of an area where the DDQ Facility will place the record. This area must be large enough to contain the largest record on the queue. For an unblocked queue, the area must be at least as large as the block size of the DDQ data set. Note that the record data (if not an Intercomm message with a formatted header) has the 4-byte length field prefix created for the QWRITE of the record.

- qnn (optional) is used to provide a specific read start point. This option may not be used for single-retrieval queues which must be read sequentially from the beginning of the queue. The type of start point depends on byte 2 of the QSW:

  If byte 2 of the QSW is not a C'L', qnn is the address of a fullword (four bytes) on a fullword boundary which contains the relative record number (RRN) of the record to be read. The RRN is a binary number corresponding to the record's position within a queue. This feature can only be used with unblocked queues that have no records larger than the DDQ data set block size. This feature is useful for random retrieval of queue records when the relative location is known. Sequential retrieval can continue from this point if this parameter is omitted on subsequent calls to QREAD.

  If byte 2 of the QSW contains a C'L', qnn is the address of a fullword on a fullword boundary that contains a specific record location originally obtained in the write feedback field of a QWRITE. Supplying this field also causes subsequent sequential retrieval to begin at this point. This feature may be used with blocked or unblocked queues.

2.9 UPDATING A QUEUE RECORD (QREADX/QWRITEX)

Updating is performed by paired QREADX/QWRITEX calls. QREADX is identical to QREAD, except that the DDQ Facility prepares to update the record if the next call is a QWRITEX. If it is not, the QREADX is treated as a QREAD. If a QWRITEX is issued, the updated record (or block, if queues are blocked) is immediately written. The length of the record must not be changed. The parameters for QREADX and QWRITEX are as follows:

```
QREADX:  qlb,qsw,qmm[,qnn]
QWRITEX: qlb,qsw,qmm
```

where the parameters are identical to those supplied on QREAD or QWRITE calls, except that an additional return code, C'6', is defined for QWRITEX. This indicates an invalid update, either because the record length was changed or because the previous call was not to QREADX.

2.10 CREATING A FRONT END CONTROL MESSAGE (FECMDDQ)

2.10.1 Messages Transmitted to a Terminal via the Front End

This facility enables the sending to the Front End of a DDQ containing messages to be transmitted to a terminal. This technique avoids interleaving of messages queued for the terminal and thus simplifies handling of groups of related output messages for the same terminal by putting them in a DDQ.

During creation of a Front End Control Message, the following actions occur:

1. The subsystem builds a permanent or semi-permanent DDQ containing messages preformatted with the control characters necessary for the destination terminal. When this DDQ is closed, byte 2 of the QSW must be set to a blank, indicating the DDQ is to be saved. The DDQ may also contain FECMs for other DDQs, or feedback FECMs mixed in with output messages. If the DDQ is prepared for a broadcast terminal group, all terminals must recognize the same message control characters. Final output message formatting is performed when each message is retrieved from the DDQ for transmission to the terminal (STCHAR insertion, OUT3270 processing if VMI not X'67', etc.).

2. The DDQ FECM is formatted by calling the service routine FECMDDQ, and initializing the message header, as described in the Intercomm Programmers Guides.

3. The DDQ is passed to the Front End by calling FESEND (FESENDC) with a VMI of X'57' in the FECM message header. If the return code from the call to FESEND(C) is not 0 (and not recoverable), then the DDQ must be freed by the caller (call QOPEN, then call QCLOSE with a C'F' in byte 2 of the QSW).

Nesting of DDQs is possible if the FECM is written to another DDQ, which in turn is passed to the terminal, or is itself nested, that is, the FECM for the second DDQ is written to a third DDQ.

2.10.2 Coding Techniques for Calls to FECMDDQ

The REENTSBS code (COBOL COBREENT and PL/1 PMIPL1 calls) for FECMDDQ is 31. The FECM is created in a user-supplied area of storage (COBOL, PL/1 or Assembler Language) or in an area obtained by FECMDDQ (Assembler Language only).

The parameter list for FECMDDQ is:

status,area,qid[,disp]

where:

- status (required) is the address of a fullword on a fullword boundary which contains the status word; on return, the first byte is set to C'0', unless an attempt to acquire an area for the FECM failed; then it is set to C'8' (same value returned in RI5). The second through fourth bytes are reserved.

- area (required) is the address of the user-supplied 112-byte FECM message area; or 0 if FECMDDQ acquires the needed area and then returns its address here.

- qid (required) is the address of the DDQ QID (first 16 bytes of QLB).

- disp (optional) parameter is the address of a one-byte area that indicates the disposition of the DDQ when it is closed by the Front End after all messages are transmitted. If it is to be saved, set to C'S'; if to be freed, set to C'F'. If this parameter is omitted, then C'F' is implied. A save disposition would be necessary for 'canned' messages to be transmitted periodically, for example, and if the terminal-id placed in the message header is the name of a broadcast group.

On return, the FECM MSGHLEN and text data fields are set; the caller is responsible for the remainder of the message header. If a user area was supplied, bytes 3 through 42 are untouched by FECMDDQ. Typically, the user will copy an input message header to the user-supplied FECM area prior to the call to FECMDDQ (and then set the VMI to X'57' and set the sending subsystem codes).

If a partially transmitted DDQ is processed by Intercomm restart, all messages on the queue will be retransmitted. For restart of nested queues to be successful, all inner queues must have a save disposition. Transmission restart, if a terminal is put down (operator request or I/O error), is from the beginning if VTAM or is controlled by the BTERM parameter DDQRSRT.

If a broadcast terminal-id is put in the message header of the FECM, it will not be logged (F2). Instead, a FECM for each terminal in the group is logged (with the real terminal-id) and queued for the corresponding terminal by FESEND processing. A 'disp' option of S must be used.

Coding techniques for calls to FECMDDQ are described in the COBOL Programmers Guide, the PL/1 Programmers Guide and the Assembler Language Programmers Guide.

2.11 SUBSYSTEM DESIGN RESTRICTIONS

When designing application subsystems that are to use the Dynamic Data Queuing Facility, the following restrictions should be observed:

- Only transient queues may be designated for single-retrieval. Neither updating nor direct retrieval (via record location or RRN) functions may be performed on these queues. A second restriction of single-retrieval queues is that records may not be written to the head of the queue.

- A subsystem which builds, or adds to, a semi-permanent or permanent queue must call QCLOSE to save it, before trying to read from the queue. Any records written to the head of a queue by a thread cannot be retrieved by the same thread without an intervening call to QCLOSE.

- DDQs are always restored to their state at time of abend (if closed), rather than their state at checkpoint time. Subsystems which reprocess messages on restart of Intercomm (data base update subsystems, file update subsystems), and which use non-transient DDQs, may have to take steps, depending on their manner of queue usage, to bypass reprocessing of their DDQs, or to delete an existing queue, and then recreate it.

- When a QCLOSE is issued for a DDQ, the queue is considered closed, and if restart occurs, it is restored to that point. However, subsystems that have issued a QCLOSE may be restarted, as they may not have finished message processing at time of abend. It is advisable to code all QCLOSE calls as near the end of application subsystem logic as possible (just before RTNLINK, GOBACK, or RETURN statements) to avoid duplicate creation of queues by a reprocessed message. If the subsystem specified its own QID, then subsystem restart logic can test for the existence of a DDQ with a specific QID (on return from call to QBUILD) and take appropriate action.

- Updating DDQs is not recommended, since DDQ applies all updates immediately. On restart, it is possible for a subsystem which had updated a record prior to the abend to find that record already updated. Precautions against this occurrence should be taken by restricting updates.

- A reading by record location changes the position for subsequent sequential retrieval each time a record location is supplied. (This is identical to the SETL function in IBM's QISAM.) The record location must not be modified in any manner (it was supplied in the feedback field of a QWRITE). Adding 1 to the record location will not supply the next record, but a QREAD without record location will do so.

- Queues to be passed to another subsystem in the same region should only be transient queues, while queues to be passed to a subsystem (or terminal) in another region may not be transient queues.

- It is important that the user have a large enough I/O area for retrieving records from a DDQ; at QOPEN time DDQ supplies the application program with the size of either the largest record on the queue (if the queue is blocked or if the largest record is greater than the physical block size of the queuing data set), or with the queuing data set's block size (if queue is unblocked). This value must be the minimum size of the I/O area supplied for QREADs. If too small an area is used, core will be overlaid.

The following chart summarizes the state of a DDQ at restart time (applicable only to semi-permanent or permanent DDQs) for various prior combinations of activity by a message processing thread.

Subsystem Activity Prior to Failure	**DDQ Status at Restart**
QOPEN + QREAD + QCLOSE	All additions/updates intact
QBUILD + QWRITE + QCLOSE	All additions/updates intact
QBUILD + QWRITE (no QCLOSE)	Queue not created
QOPEN + QWRITE (no QCLOSE)	Additions not applied (unless "T" CLOSE performed before failure)
QOPEN + QREADX + QWRITEX (no QCLOSE)	Update applied

Chapter 3

OTHER DDQ FEATURES

3.1 THE DDQINTFC MODULE

3.1.1 Automatic Queue Input Facility

The DDQINTFC module is a generalized interface to the DDQ Facility. Its function is to determine if a dynamic transient queue has been passed to a subsystem and, if so, to read the queue records and pass them to the subsystem. DDQINTFC assumes that the queue records consist of Intercomm messages. The queue record is passed to the subsystem in an identical manner as any other message; that is, the address of the queue record will be the first word of the standard parameter list passed by the Subsystem Controller.

The primary purpose of this feature is to enable the application program to process a series of messages passed from one subsystem to another subsystem in a DDQ without DDQ processing application program code in the receiving subsystem.

This facility is not supported for any COBOL subsystems, nor for any subsystems which may be dynamically loaded above the 16M line. Such subsystems must contain DDQ processing logic for a QID which may be queued to the subsystem in the input message text. In that input message, MSGHVMI is set to X'EE' to indicate to the receiving subsystem the contents of the message text. The DDQ type must be semi-permanent and the receiving subsystem must free the DDQ when it is no longer needed.

3.1.2 Operation of DDQINTFC

DDQINTFC intercepts and checks a message to see if a VMI of X'EE' is present. If not, control is passed directly to the subsystem; otherwise DDQINTFC issues a QOPEN, indicating the presence of the VMI X'EE' message. If the QOPEN is successful, it then reads a queue record and places the address of the record in the standard parameter list, replacing the original message address. It then calls the subsystem, passing it the standard Subsystem Controller parameter list. When the subsystem has processed the message, DDQINTFC will retrieve each additional message and pass it to the subsystem until the dynamic queue is exhausted, at which point it returns to the Subsystem Controller.

3.1.3 DDQINTFC and the SCT

The SCT for the subsystem using DDQINTFC defines a unique pseudo-entry point within DDQINTFC which receives all input messages.

The actual subsystem entry is identified by a unique pseudo-exit point within DDQINTFC. Since DDQINTFC may be used by several different subsystems, multiple copies of the module will be required (one with each subsystem). The following linkage editor CHANGE cards are used to define the pseudo-entry and pseudo-exit:

```
INCLUDE SYSLIB(subsystem)

CHANGE DDQENTRY(pseudo-entry)

CHANGE DDQEXIT(pseudo-exit)

INCLUDE SYSLIB(DDQINTFC)
```

3.1.4 Required Implementation Steps

When a user subsystem is to receive automatic input from a DDQ, the following steps are required for implementation:

1. Code an SCT (SYCTTBL macro), defining the subsystem code and the pseudo-entry point for DDQINTFC as the subsystem entry.

2. Use the following linkage editor control cards for resident subsystems:

   ```
   INCLUDE SYSLIB(user-subsystem)

   CHANGE DDQENTRY(SCT-pseudo-entry)

   CHANGE DDQEXIT(subsystem-entry)

   INCLUDE SYSLIB(DDQINTFC)
   ```

The Subsystem Controller then passes control to DDQINTFC as the SCT pseudo-entry point, which checks for a VMI X'EE' message and automatically retrieves messages from the associated DDQ and passes them to the subsystem for processing.

3.1.5 Using DDQINTFC with Dynamically Loaded Subsystems

Using DDQINTFC with dynamically loaded 24-Amode subsystems necessitates different linkedit requirements. A copy of DDQINTFC is part of the subsystem load module; linkage editor control statements depend on the language used.

For Assembler Language, use linkage editor statements as follows:

```
INCLUDE SYSLIB(INTLOAD)              must be first, or omit if dynamic linkedit used

INCLUDE SYSLIB(user-subsystem)

CHANGE DDQEXIT(subsystem-entry)

INCLUDE SYSLIB(DDQINTFC)

ENTRY DDQENTRY                       required control card
```

For PL/1 dynamic load, use linkage editor statements in this order:

```
INCLUDE SYSLIB(PLIV)

ENTRY PLIV

INCLUDE SYSLIB(INTLOAD)              omit if only PMIPL1 called

INCLUDE SYSLIB(user-subsystem)

CHANGE PL1CSECT(PLIMAIN)

CHANGE DDQEXIT(subsystem-entry)

INCLUDE SYSLIB(DDQINTFC)
```

3.1.6 Using DDQINTFC with the Output Utility

To use DDQINTFC for the Output Utility to process segmented messages (a series of related messages destined for one terminal), perform the following sequence:

1. Change the SCT entry for the Output Utility segmented messages SYCTTBL (Subsystem Code V is the Intercomm-supplied standard) to reflect a unique entry point, that is, SBSP=SEGOUTPT.

2. Set the &DDQBACK global in SETGLOBE to 1. Reassemble and linkedit PMIOUTPT.

3. Ensure that the following control cards are in the Intercomm linkedit (see above if the Output Utility is dynamically loaded).

```
INCLUDE SYSLIB(PMIOUTPT)

CHANGE DDQENTRY(SEGOUTPT)

CHANGE DDQEXIT(PMIOUTPT)

INCLUDE SYSLIB(DDQINTFC)
```

This produces a CSECT name SEGOUTPT, which will receive control from the Subsystem Controller and call PMIOUTPT (the Output Utility entry point) after reading a message from the DDQ, if a DDQ was passed, or simply branch to PMIOUTPT in a normal fashion when no DDQ is present.

3.1.7 Restrictions of DDQINTFC

Two restrictions exist when using this facility:

- DDQINTFC will return to the Subsystem Controller before the queue is empty if the subsystem return code is not zero, passing that nonzero return code back to the Subsystem Controller. When this occurs, the DDQ is freed and no further messages are processed.

- When DDQINTFC is used with a high-level language subsystem (PL/1), pre-editing by the Edit Utility of messages from the DDQ is unavailable.

3.2 SEGMENTED INPUT MESSAGES

3.2.1 Automatic Queuing of Segmented Input Messages

Segmented input messages from start-stop dial-up BTAM terminals and GFE and Extended TCAM terminals may be automatically queued by Intercomm until the message is complete, then passed to the subsystem. This is accomplished by the DDQ Facility in conjunction with the BSEGMOD BTAM interface module (if included in the Intercomm linkedit).

The advantages of handling segmented input in this manner are as follows:

- No special code (calls to GETSEG) is needed in the application subsystems.
- Segmented messages processed by one subsystem are fully multithreaded.
- If I/O errors cause lost segments, the handling and disposition of segments is automatic.
- There is no possibility of a subsystem receiving only parts of a segmented message.

3.2.2 Implementation Steps

Implementation of this feature requires the following steps:

1. A default queuing data set must be defined in the DDQ Data Set Table, DDQDSTBL.
2. &DDQ global must be set to 1 in the SETENV member, and the following must subsequently be reassembled:

   BLHIN

   BDIAL

   BTSEARCH

   TPUMSG
3. Include BSEGMOD in the Intercomm linkedit.

3.2.3 Restrictions

This feature may not be used if &BSCSGMT is 1 (for 2770 terminals) in SETENV, or if any CPU-to-CPU BTERM specified BSCSGMT=YES in the associated BDEVICE. This feature is not supported by the VTAM Front End.

Chapter 4

OPERATIONAL CONSIDERATIONS

4.1 INTRODUCTION

In describing operational considerations of the Dynamic Data Queuing Facility, this chapter documents the following subjects:

- Data Set Table (DDQDSTBL)
- Core requirements
- Performance considerations
- Statistics on DDQ execution and DDQ data sets
- Conditional assemblies (DDQENV)
- Intercomm linkedit requirements
- Batch program linkedit requirements
- Execution JCL and required data sets
- Restart considerations
- Using DDQ for MMU formatted output messages

For description of DDQ messages, snaps and abends, see Messages and Codes.

4.2 DATA SET TABLE

To use the Dynamic Data Queuing Facility, the DDQ Data Set Table (DDQDSTBL) must be coded. This table defines all the data sets on which queues may be created. The table is built by coding a DDQDS macro for each data set. (For description and illustration of the DDQDS macro, see Appendix A.) A sample Data Set Table is provided as member DDQDSTBL on the released SYMREL. The CSECT name is generated by the first DDQDS macro; no PMISTOP macro is required.

4.3 CORE REQUIREMENTS

The DDQMOD module consists of three CSECTs, which require about 9K. The DDQSTART module is used to initialize the DDQ feature and requires 3K. The size of the DDQDSTBL depends on the number of data sets. A representative size, assuming three data sets, would be about 260 bytes.

The other fixed core requirement is the Free Extents Tables (FET). There is one FET in core per data set not shared; the size of the FET is determined by the FETSIZE parameter of the DDQDS macro for the data set. Assuming that two out of the three data sets defined in a DDQDSTBL are not shared and use the default FETSIZE value of 600 bytes, the core requirement is 1200 bytes. However, should it become full, the FET may be dynamically expanded during execution.

The dynamic core requirements of DDQ per thread are:

QLB--User-supplied control block	48 bytes
QCB--Queue Control Block used by DDQ	136 bytes

Additionally, for a blocked queue being read, a read buffer equal to the block size of the queuing data set is also acquired. For writing to a blocked queue, the write buffer is expanded until full.

During calls to DDQ functions, a 256-byte work area is acquired by DDQMOD. It is freed prior to returning to the calling program.

4.4 PERFORMANCE CONSIDERATIONS

Transient dynamic data queues are more efficient than semi-permanent or permanent queues because of the extra I/O involved in maintaining the Queue Control Blocks. Unless otherwise required, queues should be defined as transient.

4.5 DDQ STATISTICS

Statistics on Dynamic Data Queuing Facility activities may be acquired via the System Accounting and Measurement Facility (SAM) described in the Operating Reference Manual. I/O activity for DDQ data sets, and for the QCF and SCF files (if defined) are given in the File Handler Statistics written to the SYSPRINT SYSOUT data set (see Operating Reference Manual).

4.6 CONDITIONAL ASSEMBLIES (DDQENV)

The DDQ modules DDQSTART and DDQMOD may be conditionally assembled to effect a core saving and to add or delete certain capabilities. The globals involved are defined and set in the member DDQENV. Their default values (as supplied in DDQENV on the released SYMREL), their definitions, and possible settings are described in Appendix B.

The DDQENV member as released is:

```
*              DDQ GLOBAL DECLARATIONS
*
               GBLA   &EXTNUM    NO. OF EXTENTS TO BE ALLOWED-MAXIMUM
               GBLA   &MAXLEN    MAXIMUM RECORD SIZE ALLOWED ON QUEUES
               GBLB   &UPDAT     UPDATE TO DYNAMIC QUEUES ALLOWED
               GBLB   &WRTHEAD   WRITING TO THE HEAD OF A QUEUE IS OK
               GBLB   &TRONLY    IF ONLY SINGLE-RETRIEVAL TRANSIENT Q
               GBLB   &SHARED    IF QUEUING DATA SETS ARE SHARED
               GBLB   &INTLOCK   I/O IS TO BE EXCLUSIVE ONLY
*
*              CURRENT DDQ OPTIONS IN EFFECT
*
&MAXLEN        SETA   32760      32,760 BYTES IS CURRENT MAX. RECSIZE
&EXTNUM        SETA   16         16 EXTENTS IS MAXIMUM
&UPDAT         SETB   1          UPDATING ALLOWED
&WRTHEAD       SETB   1          WRITING TO THE HEAD OF QUEUE IS OK
&TRONLY        SETB   0          ALLOW PERMANENT TYPE QUEUES
&SHARED        SETB   1          SHARED QUEUING DATA SETS SUPPORTED
&INTLOCK       SETB   1          I/O LOCKOUT IS NECESSARY
```

4.7 INTERCOMM LINKEDIT REQUIREMENTS

When using the Dynamic Data Queuing Facility with Intercomm, &DDQ SETB 1 in the SETENV member causes the ICOMLINK macro to automatically generate the required linkage editor statements for DDQ. DDQ modules are automatically included in the linkedit (even though &DDQ is 0) if either DDQ=YES, BACKOUT=YES, or the Multiregion Facility for the control region, is requested for the ICOMLINK generation of linkedit control statements. Also ensure that PMIGETNB is included in the linkedit. DDQMOD is eligible for residence in the Link Pack Area (see Link Pack Facility in the Operating Reference Manual).

Or, the following control statements may be used:

```
INCLUDE    SYSLIB(DDQDSTBL)      DDQ Data Set Table
INCLUDE    SYSLIB(DDQMOD)        DDQ Processing
INCLUDE    SYSLIB(DDQSTART)      DDQ Startup Processing
```

4.8 BATCH PROGRAM LINKEDIT REQUIREMENTS

For batch program use of DDQ, several Intercomm modules (including the File Handler) are used. The following linkedit control statements are required in the order listed:

ENTRY	DDQBATCH	(entry point in DDQSTART)
INCLUDE	SYSLIB(batch)	user's batch program
CHANGE	DDQENTRY(*batch*)	*batch* is entry point to the batch program
INCLUDE	SYSLIB(DDQSTART)	
INCLUDE	SYSLIB(DDQMOD)	
INCLUDE	SYSLIB(IXFHND00)	
INCLUDE	SYSLIB(IXFHND01)	
INCLUDE	SYSLIB(PMINQDEQ)	if any DDQ added, updated or deleted
INCLUDE	SYSLIB(BATCHPAK)	
INCLUDE	SYSLIB(PMIGETNB)	
INCLUDE	SYSLIB(DDQDSTBL)	(user version)

The QCF, SCF, and all DDQ data sets must have DISP=SHR if the batch program is executed while the on-line system is running (must not be executed during Intercomm startup). Inclusion of PMINQDEQ to serialize changes while the on-line system is running is only available under Releases 10 and 11.

4.9 EXECUTION JCL AND REQUIRED DATA SETS

All DDQ data sets must have a block size that is a multiple of eight. The following data sets are required for both the Intercomm system and batch programs using DDQ:

- All queue data sets defined by the Data Set Table (DDQDSTBL) must be BDAM data sets preformatted by the Intercomm CREATEGF utility program. They may have any valid block size (which must be a multiple of eight) and any number of blocks. When calculating the block size, note that every record in the block is proceeded by a 4-byte RDW (if not a message with a formatted Intercomm header) and treated as a variable length record. To calculate the number of blocks needed, first multiply the value coded for the BLOCKS parameter on the associated DDQDS macro by the value coded for &EXTNUM in the DDQENV table. Then multiply the result by the anticipated number of DDQs to be created on the data set. See also the description of the qmm parameter for the call to QBUILD in Chapter 2, and MMU considerations in Section 4.11.

  If a data set defined in DDQDSTBL does not have a DD statement, or is not properly formatted, the DDQSTART module will issue an indicative message and the data set will not be available for queue creation or retrieval.

Standard Intercomm File Handler DD statement requirements apply to dynamic data queues, as follows (where ddname matches that coded on the corresponding DDQDS macro):

//ddname DD DSN=dsname,DISP=SHR,DCB=(DSORG=DA,OPTCD=RF)

- If semi-permanent or permanent queues are to be created or retrieved, a data set known as the Queue Control File (QCF) must be defined. This is a keyed BDAM data set that must be preformatted by an off-line utility, KEYCREAT.

The JCL to format the QCF is as follows:

```
//XXX        EXEC  PGM=KEYCREAT,PARM=nnn
//STEPLIB    DD    DSN=INT.MODREL,DISP=SHR
//INTKEYFL   DD    DSN=anyname,DISP=(,CATLG),
//                 SPACE=(120,(nnn)),UNIT=SYSDA,VOL=SER=xxxxxx,
//                 DCB=(RECFM=F,BLKSIZE=120,KEYLEN=16,DSORG=DA)
```

where nnn in the PARM and SPACE fields specifies the number of records that are being formatted; this value is determined by the maximum number of permanent and semi-permanent queues to be created on all DDQ data sets (be generous). The BLKSIZE is a function of the value coded for &EXTNUM in the DDQENV table, as described in Appendix B. KEYCREAT formats the data set with the ddname INTKEYFL.

The following DD statement for the QCF is required at execution time in all regions using non-transient DDQs:

```
//PMIQCFDD  DD DCB=(DSORG=DA,OPTCD=RFE,LIMCT=1),
//             DSN=name,DISP=SHR
```

For subsequent executions of Intercomm, the LIMCT sub-parameter may have to be set to a new value in order to restrict or expand the search for a queue control record (one for each semi-permanent and permanent queue on each DDQ data set).

- If any queuing data sets defined in the DDQDSTBL are shared by on-line and batch update programs, or by multiple Intercomm regions, that is, SHARED=YES was coded in the DDQDS macro for the data set, then a Space Control File (SCF) data set must be defined. The SCF is a BDAM data set preformatted by the CREATEGF utility. The block size of the data set must equal the FETSIZE parameter value of the DDQDS macro for every shared queue data set; all shared queue data sets must have the same FETSIZE value. The number of blocks to be formatted must be equal to or greater than the number of shared queue data sets.

The following DD statement is required for the Space Control File:

//PMISCFDD DD DCB=(DSORG=DA,OPTCD=RF),DSN=name,DISP=SHR

In a Multiregion environment, the DDQDSTBL must contain entries in the same order for every DDQ data set used in any region in the system, regardless of whether or not a particular region uses that data set. However, the JCL for a specific region should contain DD statements only for those data sets that it actually accesses. (See example below.) This causes WTO messages to be issued at the startup of the region, stating that the DD statements for the data sets which are not used in the region are missing, but causes no other problems. Any DD statement not present in the JCL is simply inaccessible to the region. (The inclusion of all ddnames in the DDQDSTBL is necessary because of the way DDQSTART processes the SCF file.) Also, entries for shared queue data sets must be coded before entries for dedicated queue data sets.

For example: DDQ1, DDQ2 and DDQ3 are ddnames for three DDQ data sets. DDQ1 is shared between control and satellite regions, but DDQ2 and DDQ3 are used only by the satellite region, that is, they are dedicated.

The DDQDS macro coding for the DDQDSTBL is as follows (the identical table must be linkedited into each region):

```
                 DDQDS  DDNAME=DDQ1,SHARED=YES,...  (other parms
                 DDQDS  DDNAME=DDQ2,SHARED=NO,...   as required)
                 DDQDS  DDNAME=DDQ3,SHARED=NO,...
```

The Control Region JCL is as follows:

```
//CR            EXEC ...
//PMIQCFDD      DD  DISP=SHR,DSN=...
//PMISCFDD      DD  DISP=SHR,DSN=...
//DDQ1          DD  DISP=SHR,DSN=...
```

The Satellite Region JCL is as follows:

```
//SR            EXEC ...
//PMIQCFDD      DD  DISP=SHR,DSN=...
//PMISCFDD      DD  DISP=SHR,DSN=...
//DDQ1          DD  DISP=SHR,DSN=...
//DDQ2          DD  DSN=...                    DISP=SHR recommended
//DDQ3          DD  DSN=...                    DISP=SHR recommended
```

To prevent the start of any region from wiping out a queue still being used by another region, semi-permanent queues should not be kept on a shared data set (see below). DDQ data sets (ddname=THREDLOG) used for the Backout-on-the-Fly facility (see the File Recovery Users Guide) may not be shared across regions. Under Release 9 only, Backout-on-the-Fly must be confined to one region if there are any shared DDQ data sets.

When a new DDQ data set is created, a new entry must be added to the DDQDSTBL. (If shared it must immediately precede the dedicated entries, if not it must follow the dedicated entries.) The DD statement for the data set must be placed in the JCL for only that region(s) that accesses it. Also, if the new DDQ data set is shared, the SCF BDAM data set must be recreated to add a block for the new DDQ data set (if the current number of blocks is less than the new total number of shared DDQ data sets). The QCF data set must also be recreated if the SCF file is recreated.

Batch programs always require both the Queue Control File and Space Control File. Batch programs using DDQ must not be executing while Intercomm is starting up. They may only execute if Intercomm is down or if startup has completed. This is because DDQ initialization closes all open queues. If a batch program using DDQ abends, any queue opened and not closed by the batch program will be inaccessible until an Intercomm startup (normal or restart) takes place.

If a DDQ data set containing permanent or semi-permanent queues is scratched and recreated, then the QCF data set (and SCF data set, if used) must also be recreated.

If it is necessary to change the FETSIZE value for any shared DDQ data set, then every DDQDSTBL entry for a shared DDQ data set must be correspondingly changed to have the new FETSIZE value. The QCF file must be recreated, as well as the SCF file (which must specify the new FETSIZE as the BLKSIZE for that recreation).

4.10 RESTART CONSIDERATIONS

DDQ includes its own logic for preservation of non-transient queues at system startup time. This capability is independent of the Intercomm File Recovery Special Feature.

Preservation of DDQs depends upon the queue being closed at the time of job termination, normal or abnormal. Depending on the mode of execution and type of queue, a DDQ existing in a previous execution may or may not be available at subsequent executions.

A permanent queue is available, if closed, at startup and restart. A semi-permanent queue is not available at startup, but is available at restart if previously closed with any option but free the queue, and RESTART=YES must have been coded on the DDQDS macro for the associated DDQ data set. A transient queue is never available at startup or restart.

A queue is preserved, not restored; it is unavailable at restart time if it was not previously closed by the program creating the queue. Refer to Section 2.11, "Subsystem Design Restrictions," for additional considerations of restart processing.

After an Intercomm (Control Region) restart with message restart, a DDQ which has been passed to the Front End by a FECM will be completely retransmitted if Front End processing of the DDQ did not complete in its previous execution, and if RESTART=YES is defined for the receiving terminal. If a FECM DDQ is created in a Satellite Region, and that region abends before the DDQ is completely transmitted by the Control Region, then the satellite region must be restarted in restart mode so that the queue is preserved and Control Region transmission can continue if already in progress, or begin when the previously queued FECM for the terminal is dequeued for transmission processing. Code RESTRNL for the Intercomm execution parameter if message restart is not used in the Satellite Region.

If the File Recovery facility is in use, DDQs must not be specified for file reversal. To recover DDQs, the queuing data sets, the Queue Control File, and the Space Control File (if any), must be defined by FAR entries specifying RECREATE. This causes additional overhead on writes to DDQs, as all writes are logged, and should only be used to protect DDQs against disk head crashes, or other permanent disk I/O problems.

Do not code the REVERSE parameter on the FAR entry for any of these DDQ files. If a serious error occurs on any of the above files, they must first be recreated by the File Recovery facility off-line utility IXFCREAT, before restarting Intercomm.

4.11 USING DDQ WITH THE MESSAGE MAPPING UTILITIES (MMU)

If a DDQ data set is to be used for DDQs automatically created by the MMU output message processing routines (see Chapter 2), that data set is specified in the Message Mapping Utilities Vector Table (MMUVT). The ddname of the data set is coded for the OPMDDNM parameter of the MMUVT macro and must also be defined in the DDQ Data Set Table (DDQDSTBL). If no ddname is provided in the MMUVT, MMU uses the DDQ data set specified as the default in the DDQDSTBL.

The DDQ created via a MAPEND call option is of the semi-permanent type, therefore PERMS=YES must be coded on the DDQDS macro defining the data set for MMU DDQs. Considerations for restart affecting such queue types and FECMDDQs are described in the previous section. Either a blocked or an unblocked (see DDQDS macro) DDQ data set may be used for MMU. The minimum block size must be 2000 bytes (longest message length including header, +4).

MMU does not request blocking of messages in the DDQ, and uses the default number of blocks (messages) per extent defined on the DDQDS macro for the DDQ data set used by MMU. A maximum of one message per block is written. Therefore, the maximum number of messages that may be put in one DDQ is the value coded for the BLOCKS parameter on the associated DDQ data set, times the value coded for the &EXTNUM global in the DDQENV table. If 3270 printer output is placed in a DDQ, and the logical page size mapped for a printer exceeds the device block size (default is 1920), then more than one message (block) may be required per logical printer page. See the DVMODIFY macro description in Basic System Macros.

Chapter 5

OFF-LINE DDQ PRINT UTILITY

5.1 INTRODUCTION

An off-line utility, DDQPRT, is provided under Releases 10 and 11 to perform the following operations:

- Print the records in a specific DDQ (in EBCDIC and hexadecimal)
- Print all the DDQ's on the DDQ data set pointed to by the specified ddname (in EBCDIC and hexadecimal)
- Print information from the Queue Control File, for either a specific ddname or all DDQ data sets.

Installation of the utility will require linkediting it using the Intercomm LKEDP Procedure. The linkedit control statements are shown in Section 4.8 where DDQPRT is the batch program. The INCLUDE statement for DDQDSTBL should be changed to:

INCLUDE ddname(DDQDSTBL)

where ddname is the label of a DD statement pointing to the load library containing the user's DDQDSTBL (if not the library indicated by the Q parameter for LKEDP).

The load module name should be DDQPRINT. A prelinked DDQPRINT load module is provided on MODREL (or on MODLIB from installation job 40), however the load module must be relinked (see Section 4.8) if any of the included modules are updated or reassembled.

5.2 EXECUTION JCL

The syntax of the EXEC statement is as follows:

```
//     EXEC   PGM=DDQPRINT,PARM='{qid-name      }'
                                {ddname[,{L}]]}
                                {         {Q} }
                                {         {S} }
                                {,L           }
```

Each of the possible PARM options will provide printed output as follows:

1. PARM='qid-name' prints the DDQ records for this QID name only.

2. PARM='ddname' prints all the DDQ's that are pointed to by this ddname on the QCF file.

3. PARM='ddname,L' prints all the DDQ's that are pointed to by the ddname and lists all the QID names and ddnames found on the Queue Control File.

4. PARM='ddname,Q' prints all the DDQ's that are pointed to by the ddname and lists only the QID names on the specified data set from the Queue Control File.

5. PARM='ddname,S' lists only the QID names on the specified data set from the Queue Control File.

6. PARM=',L' lists all the QID names and ddnames found on the Queue Control File.

NOTE: ddname must be a valid 1 to 8 character DDQ data set ddname; qid-name must be exactly 16 characters.

The following DD statements must be present in the execution JCL:

//STEPLIB	DD	points to the library containing the DDQPRINT load module
//PMIQCFDD	DD	points to the user's Queue Control File with parameters: DCB=(DSORG=DA,OPTCD=RFE,LIMCT=c) *where* c is the LIMCT value used in the on-line JCL
//PMISCFDD	DD	points to the user's Space Control File with parameters: DCB=(DSORG=DA,OPTCD=RF)
//PRINTOUT	DD	a SYSOUT file with DCB=(LRECL=133,BLKSIZE=nnn)
//PRINT	DD	a SYSOUT file with DCB=(LRECL=133,BLKSIZE=nnn) *where* nnn is a multiple of 133

// plus 1 DD statement for each DDQ data set to be processed, each with parameters:
DCB=(DSORG=DA,OPTCD=RF)

The requested data is printed on the PRINT file; error messages are written to PRINTOUT.

If executed while the on-line system is running, ensure all referenced DDQ data sets and the QCF and SCF (if used) files have DISP=SHR in both the batch and on-line JCL.

5.3 EXAMPLES:

To print out a specific DDQ whose qid-name is known:

```
//            EXEC PGM=DDQPRINT,PARM='AAAAAAAAKKKKKHIJ'
//STEPLIB     DD   DSN=INT.MODLIB,DISP=SHR
//PMIQCFDD    DD   DSN=INT.PMIQCFIL,DISP=SHR,
//                 DCB=(DSORG=DA,OPTCD=RFE,LIMCT=c)
//PMISCFDD    DD   DSN=INT.PMISCFIL,DISP=SHR,
//                 DCB=(DSORG=DA,OPTCD=RF)
//MAROTH04    DD   DSN=INT.MAROTH04,DISP=SHR,DCB=(DSORG=DA,OPTCD=RF)
//PRINTOUT    DD   SYSOUT=A,DCB=LRECL=133
//PRINT       DD   SYSOUT=A,DCB=LRECL=133
```

In this example, the DDQPRINT load module is on INT.MODLIB, and the DDQ (qid-name=AAAAAAAAKKKKKHIJ) is known to be on the DDQ data set pointed to by the MAROTH04 DD statement. If the data set on which the DDQ resides is not known, DD statements for all the DDQ data sets used on-line should be added to the execution JCL.

To print out all the DDQ's on a DDQ data set pointed to by a specific ddname and list all the qid-names for the DDQ's which reside on this data set:

```
//            EXEC PGM=DDQPRINT,PARM='MAROTH02,Q'
//STEPLIB     DD   DSN=INT.MODLIB,DISP=SHR
//PMIQCFDD    DD   DSN=INT.PMIQCFIL,DISP=SHR,
//                 DCB=(DSORG=DA,OPTCD=RFE,LIMCT=c)
//PMISCFDD    DD   DSN=INT.PMISCFIL,DISP=SHR,
//                 DCB=(DSORG=DA,OPTCD=RF)
//MAROTH02    DD   DSN=INT.MAROTH02,DISP=SHR,DCB=(DSORG=DA,OPTCD=RF)
//PRINTOUT    DD   SYSOUT=A,DCB=LRECL=133
//PRINT       DD   SYSOUT=A,DCB=LRECL=133
```

Again, in this example, the DDQPRINT load module is on INT.MODLIB. The only DDQ data set DD statement required is for ddname MAROTH02, since it is the only DDQ data set ddname which will be processed.

5.4 ERROR CONDITIONS

Error messages are written out to the PRINTOUT SYSOUT data set. If storage cannot be obtained for a DDQ record, a message is written and an intentional program check is forced.

5.5 END OF JOB STATISTICS

At normal end-of-job, statistics are written out to the PRINT SYSOUT data set. The statistics give counts of the following:

1. number of Queue Control Blocks read
2. number of Queue Control Blocks listed
3. number of DDQ's processed
4. number of records read and printed from the DDQ's
5. number of errors encountered before processing was stopped, if necessary.

5.6 EXECUTION CONSIDERATIONS

This program may be executed after Intercomm startup completes provided DISP=SHR is coded on all DDQ data set DD statements (including those for PMIQCFDD and PMISCFDD), for both the batch and on-line JCL, and provided that the identical DDQDSTBL is used. Ignore error messages (DQ051I) issued because DD statements are omitted for one or more of the data sets named in the DDQDSTBL used for execution of the batch program.

If a DDQ in a file named in the batch execution JCL is updated or deleted while this program is executing, results may be unpredictable. DDPRT does not modify existing DDQ records or the QCF or SCF files, and keeps all existing queues. <u>If the on-line system is down or closes down (abends) while this program is executing, do not start/restart the on-line system until DDQPRINT ends</u>.

Appendix A

THE DDQDS MACRO

The format of the DDQDS macro is as follows:

```
(blank)    DDQDS      DDNAME=ddname

                      [,ACTIVE={NO }]
                      [        {YES}]

                      [,BLOCKNG={NO }]
                      [         {YES}]

                      [,BLOCKS={nn}]
                      [        {8 }]

                      [,DEFAULT={YES}]
                      [         {NO }]

                      [,FETSIZE={nnnnn}]
                      [         {600  }]

                      [,PERMS={YES}]
                      [       {NO }]

                      [,RESTART={NO }]
                      [         {YES}]

                      [,SHARED={YES}]
                      [        {NO }]

                      [,TRESH={nn}]
                      [       {80}]
```

ACTIVE
is coded NO if the data set is not to be used for dynamic data queuing. The default is YES.

BLOCKNG
NO specifies that no blocked queues are allowed on the data set. If NO is specified, it overrides any request for creation of a blocked queue via the QBUILD function. The default is YES.

BLOCKS
specifies the number of physical blocks to be allocated for each queue extent, unless otherwise specified at QBUILD time. Code as a decimal value from 1 to 32767. The default is 8.

DDNAME

identifies the ddname of a BDAM data set that is to be used for dynamic data queues. This parameter must be coded.

DEFAULT

is coded YES if this data set is to be the default data set. Queues will be created on this data set unless otherwise specified at QBUILD time. The default is NO.

FETSIZE

defines the initial size of the Free Extents Table (FET), which consists of 12-byte elements, one for every set of contiguous physical blocks not in use. The amount of expected fragmentation of the total queuing space should determine the FET size. When SHARED=NO is coded, the FET is core-resident and, if full, is automatically expanded by the DDQ program. If SHARED=YES, the FET is disk-resident and cannot be expanded, and FETSIZE <u>must</u> be the same value for <u>all</u> shared data sets, and is used as the BLKSIZE when creating the SCF data set (see Section 4.9). Code as a decimal value from 36 to 32760 divisible by 12. The default is 600. The formula for calculating FETSIZE is:

$$12*\frac{(\text{\# Blocks in largest shared DDQ data set})}{(\text{smallest value of BLOCKS parm for all shared DDQ data sets})}$$

PERMS

is coded YES if non-transient queues will be created on the data set. If SHARED=YES, this parameter must be coded as YES. The default is NO.

RESTART

is coded NO if semi-permanent queues are not to be recovered on Intercomm restart. The default is YES (required for shared DDQ data sets).

SHARED

is coded YES if the data set is to be shared by batch and/or multiple on-line regions. If this parameter is coded YES, the PERMS parameter must also be YES. The default is NO.

TRESH

is the percentage of the data set which must be in use before a warning message (DDQ010I) is generated. Code as a decimal value from 0 to 100. The default is 80.

Appendix B

DDQENV GLOBALS

Global	Setting	Default	Definition
&MAXLEN	SETA	32760	The maximum size a queue record may have. Code as a decimal value from 5 to 32760.
&EXTNUM	SETA	16	The number of extents a queue may own. Code as a decimal value from 1 to 44. Changing this value changes the block size of the Queue Control File: block size = (&EXTNUM*4)+56. If this value is changed, the QCF data set (and SCF data set, if used) must be scratched and recreated.
&UPDAT	SETB	1	Generate code to support the update functions, QREADX and QWRITEX.
&WRTHEAD	SETB	1	Generate code to support writing to the head of a queue.
&SHARED	SETB	1	Generate code to support shared (by different regions) queue data sets.
&TRONLY	SETB	0	Generate code to support all queue types. The largest core saving is obtained by setting on the &TRONLY global so that only support for single retrieval transient queues is generated.
&INTLOCK	SETB	1	Generate code to support single-threaded accesses to a queue. If using the DDQ Facility with the Multiregion Support Facility, the &INTLOCK global must be on. If off, multithread access to the same queue is possible.

INDEX