z/OS
2.5

Bulk Data Transfer Installation

SC14-7582-50

Note

Before using this information and the product it supports, read the information in “Notices” on page 179.

This edition applies to Version 2 Release 5 of z/OS® (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2021-09-30

Contents

Figures

About This Book

This book is a guide to installing the Bulk Data Transfer (BDT) licensed program. It describes how to plan your BDT configuration, define BDT to MVS and ACF/VTAM, allocate and format data sets, create an initialization stream, write start procedures, and write user exit routines.

Who Should Read This Book

This book is for system programmers who must install BDT, test it, and create a production system.

You should be familiar with Systems Network Architecture (SNA), ACF/VTAM, and JES2 or JES3 (depending on which job entry subsystem your site uses).

How to Use This Book

If you are new to BDT you should start with Chapter 1, “Introduction,” on page 1. It is an overview of the installation procedures covered in the rest of the book. Then you can read the rest of the book in sequence. Note that the sequence in which the chapters are presented is intended to guide you; you are not required to install in that order.

Related Reading

Where necessary, this book references information in other books, using shortened versions of the book title. For complete titles and order numbers of the books for all products that are part of z/OS, see *z/OS Information Roadmap*.

Syntax Conventions Used in This Book

The conventions used in this book to describe the syntax of initialization statements and macro instructions are:

- Any word or character shown in uppercase must be coded as it is shown.
- Any word or character shown in lowercase may be replaced by a value that you provide.
- When a vertical bar (|) separates items you may select one of the items unless the item description says you can select more. Do not code the vertical bar.
- When items are enclosed in braces ({}) you must select one of the enclosed items. Do not code the braces.
- When items are enclosed in brackets ([]) the enclosed items are optional. You may select none, some, or all of the items. Do not code the brackets.
- Default values are underlined.

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead "If you have a technical problem" on page xv.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM® Documentation function
If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at ibmdocs@us.ibm.com.

Feedback on the z/OS product documentation and content
If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS BDT Installation, SC14-7582-50
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under IBM z/OS Product Documentation Update Policy (www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/ibm-zos-doc-update-policy?OpenDocument).

Summary of changes for z/OS BDT Installation for Version 2 Release 5 (V2R5)

This information contains no technical changes for this release.

Summary of changes for z/OS BDT Installation for Version 2 Release 4 (V2R4) and its updates

This information contains no technical changes for this release.

Summary of changes for z/OS BDT Installation for Version 2 Release 3 (V2R3) and its updates

This information contains no technical changes for this release.

Chapter 1. Introduction

Bulk Data Transfer (BDT) is a licensed program that transfers data from one computer system to another. The type of data that it transfers depends on which of its features is installed:

- The File-to-File feature allows users at one z/OS JES2 or z/OS JES3 system in a SNA network to copy data sets to or from another z/OS JES2 or z/OS JES3 system in the network.
- The SNA NJE feature allows z/OS JES3 users to transmit jobs, output (SYSOUT), commands, and messages from one computer system to another within a SNA network. Any of the following systems can participate in the network: z/OS JES2, z/OS JES3 (with BDT), VM/SP (with RSCS Networking), and VSE/Advanced Functions (with VSE/POWER).

A third feature, the Base feature, is a prerequisite to the File-to-File feature and the SNA NJE feature.

This chapter presents an overview of the migration considerations and installation procedures described in this book. They are:

- Planning your migration
- Planning your configuration
- Defining BDT to MVS
- Defining BDT to VTAM
- Allocating BDT and TQI data sets
- Formatting TQI data sets
- Creating a BDT initialization stream
- Writing BDT and TQI start procedures
- Writing user exit routines.

Planning Your Configuration

There are several planning decisions you must make as a first step in installing BDT:

1. Within a JES complex, each file-to-file node must have a *global* or *local* relationship with every other file-to-file node it communicates with. (The terms global and local, as used in BDT, refer to a relationship between nodes and not to the JES3 global or local processors.) If you will have file-to-file nodes you must plan these relationships. The global-local relationship between BDT nodes determines which node processes transactions to copy data sets between the two nodes, regardless of the node at which the transactions are submitted. The global node always processes transactions.
2. The *transaction queuing integrity (TQI)* facility ensures that the commands and file-to-file transactions that users submit reach the BDT work queue. TQI also makes it possible for users at other processors to receive BDT messages. TQI is a separate program that runs in its own address space. TQI is optional. You must decide whether to use it and, if you do use it, in which processors of a JES complex it will run.
3. BDT runs in its own address space. A JES complex may have one or more BDT address spaces. Each processor in the complex may have one or more BDT address spaces. A complex with multiple BDT address spaces is called a *poly-BDT complex*. You must decide whether you want to set up such a complex. A poly-BDT complex is beneficial during testing, as a way to separate testing from production work.

Configuration planning is discussed in Chapter 2, “Planning Your Configuration,” on page 5.

Defining BDT to MVS

BDT operates under the control of MVS and therefore requires the following MVS definitions:

1. You must define BDT as a secondary MVS subsystem by creating an entry in an IEFSSN*xx* member of SYS1.PARMLIB.
2. You must specify MVS system parameters in an IEASYS*xx* member of SYS1.PARMLIB.
3. You must give authorized program facility (APF) authorization to SYS1.BDTLIB, the BDT module library, by updating SYS1.PARMLIB member IEAAPF*xx*.
4. If you are installing BDT in a JES3 complex, you will have to add several statements containing BDT-related information to the JES3 initialization stream. The statements are CONSOLE, SYSID, and, for SNA NJE customers, NJERMT.

MVS definitions are discussed in Chapter 3, “Defining BDT to MVS,” on page 15.

Defining BDT to VTAM

BDT runs as a VTAM application program, requiring several VTAM definitions:

1. You must define BDT as a VTAM application program on the APPL definition statement. One APPL statement is required for each type of node your system will have, that is, one APPL statement for a file-to-file node and one APPL statement for a SNA NJE node.
2. You must define each remote node as a VTAM cross-domain resource by coding a CDRSC definition statement for each remote node.
3. You must define session parameters to VTAM by creating VTAM logon mode table entries.

VTAM definitions are discussed in Chapter 4, “Defining BDT to VTAM,” on page 19.

Allocating BDT and TQI Data Sets

With MVS and VTAM definitions complete you can start work on BDT itself. Your first task is to allocate data sets that are used during BDT operation:

1. You must allocate a data set to contain the BDT initialization stream. Later you will put initialization statements into this data set and then specify its name on the procedure that the operator invokes to start BDT.
2. You must allocate a data set to contain the BDT work queue, in which BDT places jobs waiting to be processed.
3. If you plan to have a system generic master job definition (GMJD) library for users, you must allocate it. A system GMJD library contains precoded file-to-file transaction definitions.
4. If you want users to have ISPF panels available for submitting file-to-file transactions, you must allocate several data sets.
5. If users are to have the commands and file-to-file transactions that they submit checkpointed by TQI, you must allocate a TQI checkpoint data set. Within a JES complex, this data set must be shared by a processor that has a BDT address space and processors that have TQI address spaces.
6. You must allocate a TQI bit-map data set if you allocate a TQI checkpoint data set.
7. In order for users to receive BDT messages you must allocate one or more message data sets. Each processor of a JES complex that has a TQI address space must have a message data set shared by that processor and a processor that has a BDT address space.

Chapter 5, “Allocating BDT and TQI Data Sets,” on page 25 describes how to allocate BDT and TQI data sets.

Formatting TQI Data Sets

The TQI checkpoint data set, the TQI bit-map data set, and the message data sets must be formatted before they can be used. You must run batch jobs to format them, as described in Chapter 6, “Formatting TQI Data Sets,” on page 31.

Creating a BDT Initialization Stream

To define the environment in which BDT will operate each time it is started you must code initialization statements. You can specify the amount of main storage BDT will use, the names of nodes in the network, the pacing rate for communication between nodes, and many more parameters. You put these statements, which make up the "initialization stream", into a data set you previously allocated. Later on you identify the data set's name in the BDT start procedure so that during BDT startup the initialization stream is run, and the BDT subsystem gets initialized.

You can code initialization statements from scratch or you can modify the IBM-provided initialization statements in SYS1.SBDTSAMP.

You may wish to code several initialization streams. If several streams are available then the operator, by warm starting BDT, can easily change the network configuration or select different options.

Chapter 7, "Creating a BDT Initialization Stream," on page 33 describes how to code initialization statements.

Writing BDT and TQI Start Procedures

You must provide start procedures that the operator can use to start BDT and TQI. Starting BDT creates a BDT address space, and starting TQI creates a TQI address space.

The BDT start procedure invokes the BDT program, identifies the data sets that you previously allocated and that are used by BDT during its operation, and invokes the initialization stream that you previously created. The TQI start procedure invokes the TQI program and identifies the TQI data sets that you previously allocated and that are used by TQI during its operation.

You can code start procedures from scratch or modify the IBM-provided sample start procedures in SYS1.SBDTSAMP. Chapter 8, "Writing BDT and TQI Start Procedures," on page 61 describes start procedures.

Writing User Exit Routines

User-written exit routines are customer-coded programs that receive control at specific points during the processing of IBM programs. BDT user-written exit routines can be of two types:

- Authorization exit routines, which you can write to control who may send and receive commands and file-to-file transactions. You must code these exit routines. IBM provides samples in SYS1.SBDTSAMP.
- Customization exit routines, which you can write to alter command processing, transaction processing, message processing, and the initialization process. You can code these exit routines if you want; they are not required.

Chapter 9, "Writing User Exit Routines," on page 65 discusses the steps involved in developing user exit routines. Chapter 10, "User Exit Routine Reference," on page 81 describes the input, processing, and output expected of each routine. The routines are presented in alphanumeric order. Chapter 11, "Mapping Macro Reference," on page 139 and Chapter 12, "Executable Macro Reference," on page 143 describe the macros you can use in your routines. The macros are presented in alphabetic order.

Chapter 2. Planning Your Configuration

This chapter describes how to:

- Plan global and local relationships between file-to-file nodes. Global nodes have BDT work queues.
- Plan the use of the transaction queuing integrity (TQI) facility within a JES complex. TQI ensures that commands and file-to-file transactions reach the BDT work queue, and allows users to receive BDT messages.
- Decide whether to have a poly-BDT complex (multiple BDT address spaces within a JES complex).

Step 1. Plan Global and Local Relationships (File-to-File Customers Only)

In BDT, a node at one end of each file-to-file connection must be designated global while the other node is designated local.[1] Because a node may take part in many sessions, it may be global for some sessions and local for others. A node cannot be described as global or local by itself. A node is global or local only with respect to another node.

A global node of a given session has a queue where transactions involving the two nodes are kept until they are scheduled, executed, and finally purged. For example, if a session exists between node A and node B, and node A is the global node for that session, then transactions involving file transfers between A and B are queued and scheduled at A. This is regardless of the direction of data flow. Because A keeps the queue and does the scheduling, it may require more machine resources and more experienced operators than node B. This may be offset by the fact that B could be a global node for some other session pair.

You must designate the global and local relationship of each node by using the T=LOCAL parameter of the BDTNODE initialization statement (described on page "BDTNODE—Define Session Characteristics between Home and Remote Nodes" on page 37).

There are several ways to approach the planning of global-local relationships, as described on the following pages.

Global-local relationships in a central hub network

One possible configuration is a central hub or star arrangement, shown in Figure 1 on page 6. In this configuration a single node is selected as the hub with all sessions radiating from it. The hub node becomes the global node for all sessions.

[1] As previously mentioned, the BDT global-local terminology has nothing to do with similar terminology used by JES3.

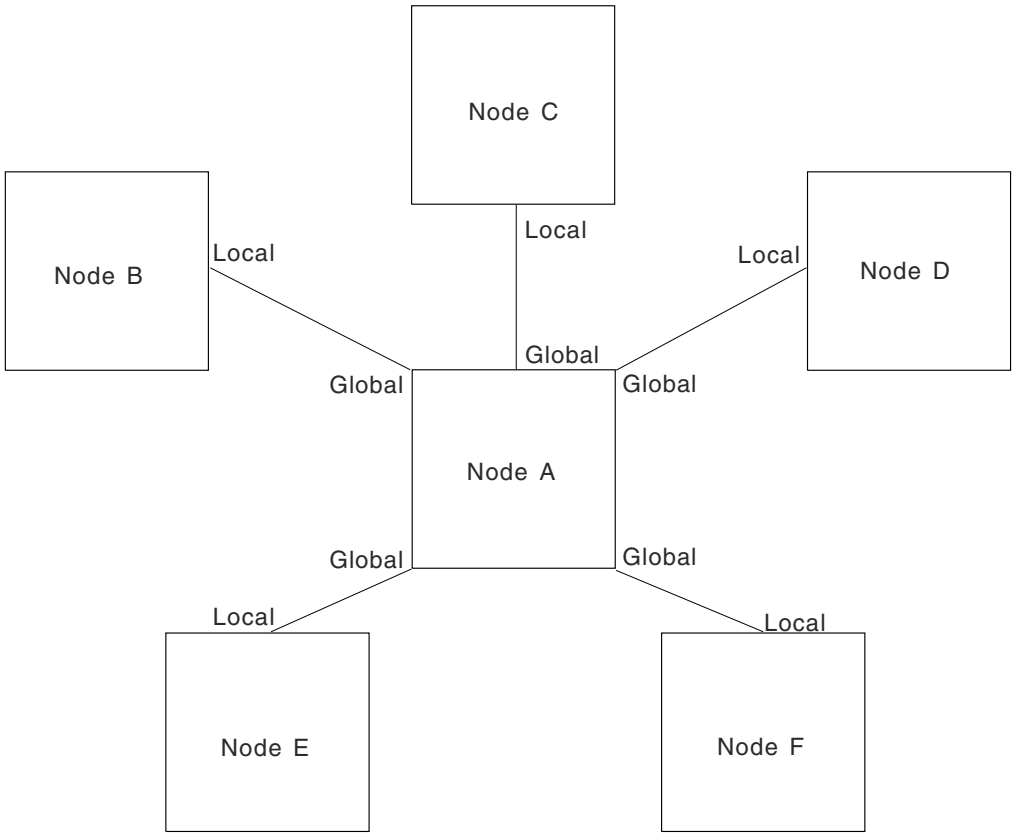


Figure 1. Example of global-local relationships in a central hub network

A central hub network has the advantage of central control of the network. However, it has several disadvantages:

- It centralizes processor utilization and may require more processor resources than are available.
- It makes it difficult to transfer data from one remote site to another. There is no direct session between remote nodes so a remote node user must first send a file to the hub and then submit a transaction at the hub to transfer the data from the hub to the local node.

Global-local relationships in a decentralized network

A second network model, shown in Figure 2 on page 7, has all nodes that will communicate connected by a session. It is possible to have one node global for all the sessions in which it participates (node B) and another node local (node C) for all its sessions; however, most nodes will be global for some sessions and local for others. This is a decentralized network. Its main benefit is that processor cycles used for networking are spread throughout the network, and it is possible to send files from one node to another with a single transaction.

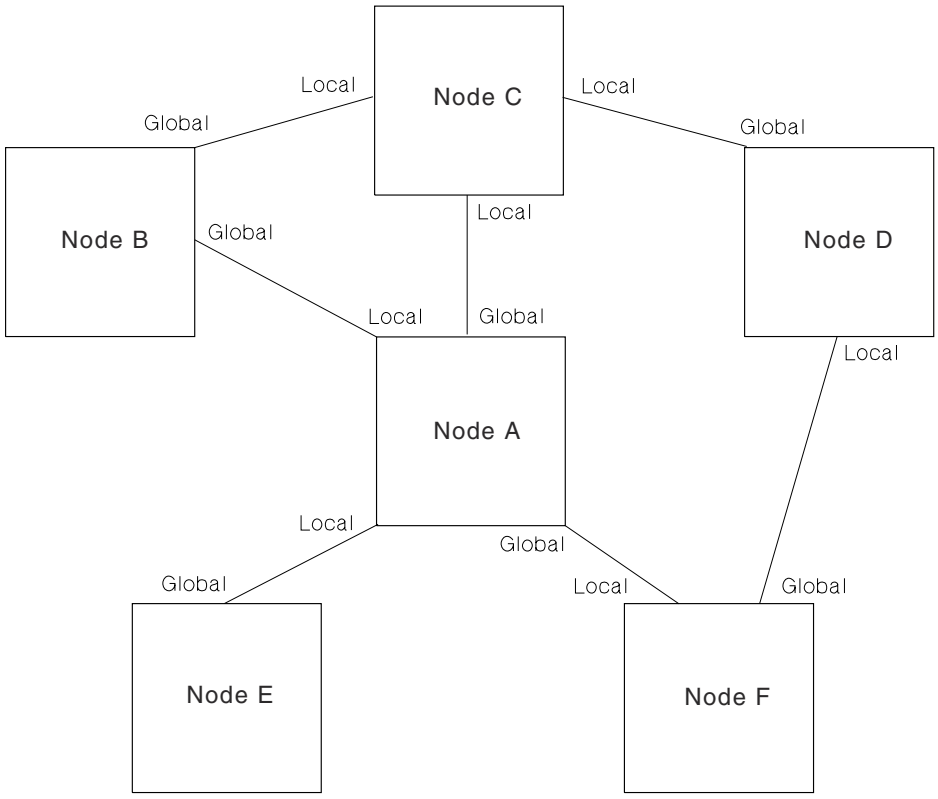


Figure 2. Example of global-local relationships in a decentralized network

In the previous discussion we have been talking about sessions, not physical lines. For example, in Figure 2 on page 7, you might have a hub arrangement for physical lines and yet define the sessions shown. In this case a transaction sent from node F to node D would pass through the NCP associated with node A. Connection definitions can be changed without modifying any physical links.

The effect of global-local relationships on users

You should consider the effect of global-local relationships on users. Submitters of file-to-file transactions must understand which node is global for each of their file transfers. For example, in Figure 3 on page 8, assume that you are a user at node A, which is global to node B but local to node C.

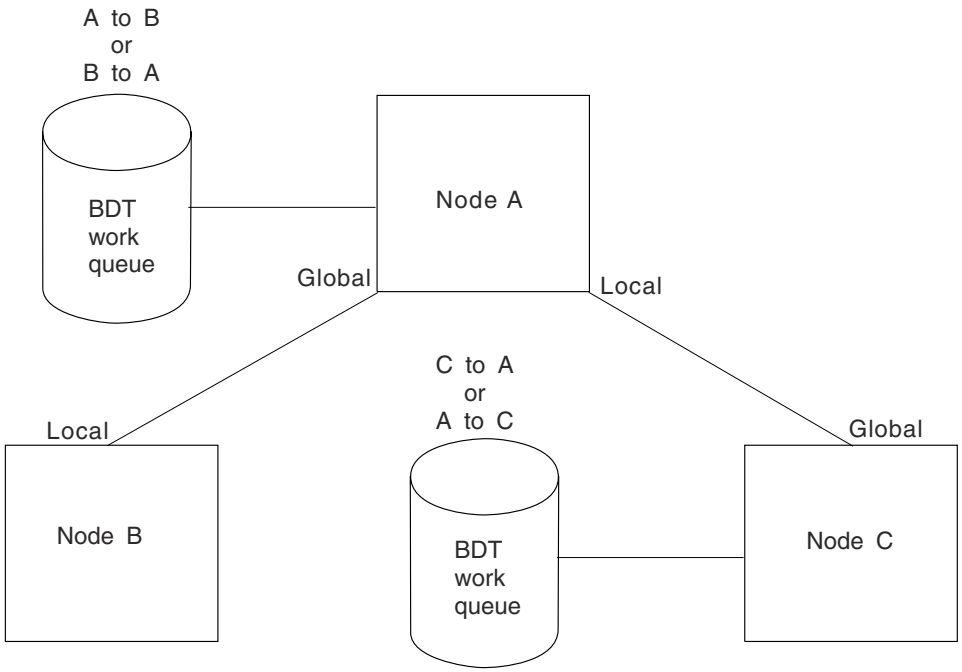


Figure 3. Global-local effects on queue location

If you send data to or fetch data from node B, your transaction will be queued and scheduled at your node, A. To inquire about that transaction's status, you would use the I,J=*n* command. If you submit a transaction to transfer a file in either direction between nodes A and C, the transaction will be queued and scheduled by node C. To inquire about the transaction you must use the SEND (T) command to send an INQUIRY (I) command to node C (T,C,I,J=*n*).

As you can see, users must understand the topology of the network in order to issue the proper command. Users at the global node for a given transaction issue the command directly. Users at the local node use the SEND (T) command to issue the command on the remote global node.

Global-local relationships also affect the use of dependent transaction control (DTC). All transactions in a DTC network must be scheduled at the same global node. DTC cannot be used to send a file to node B and then to C after the node B transaction completes. This is because the two transactions are queued at two different nodes.

Step 2. Plan the Use of TQI

You must decide whether to use the transaction queuing integrity (TQI) facility and, if you do use it, on which processors of a JES complex you will use it.

What TQI Does

Each time a user submits a request (command or file-to-file transaction) to BDT, the request travels from the user address space to the TQI address space to the BDT address space on the global (scheduling) node. From there, BDT writes the request onto the BDT work queue. For example, if a user submits a request through a TSO terminal, the request travels from the TSO address space to the TQI address space to the BDT address space. The user and TQI address spaces may be in the same processor as the BDT address space or in different processors within the same JES complex. Thus, the request has to travel from one address space to another and possibly from one processor to another. The actual path that a request follows depends on the configuration of your installation and whether your node is the global (scheduling) node.

TQI Prevents Loss of User Requests

The more address spaces and processors in the path of a request the greater the chance of losing the request. A lost request is one that does not reach the BDT work queue on the global (scheduling) node. A request would be lost, for example, if one of the address spaces or processors in its path were disabled. Users must resubmit lost requests.

TQI protects against the loss of requests by creating a checkpoint record of each request. TQI retains the checkpoint record until the request has been written onto the BDT work queue. If a request is lost after the checkpoint record has been created, BDT recovers the request from the checkpoint record.

TQI Routes BDT Messages to Users

Each command or file-to-file transaction that a user submits results in BDT issuing one or more messages to the user. When there are a large number of users, the message traffic can become heavy. BDT stores these messages on a data set. TQI periodically reads messages from the data set and routes them to the user. This method of storing and routing messages eliminates a potential burden on virtual storage by eliminating the need to store BDT messages there.

The data sets that TQI uses

TQI uses three different types of data sets:

- To record user requests, TQI uses the *TQI checkpoint data set* and the *TQI bit-map data set*. The checkpoint data set contains the requests and the bit-map data set contains control information. Both data sets must be accessible to the BDT address space and to each TQI address space in a complex.
- To store BDT messages, TQI uses *message data sets*. There must be a separate message data set that is accessible to each TQI address space. All message data sets must be accessible to the BDT address space.

  Note: A message data set is optional for any processor that is going to accept only intra-node file-to-file transactions.

Figure 4 on page 10 shows a three-processor JES complex that has one BDT address space and two TQI address spaces. The checkpoint data set and the bit-map data set are accessible to all of the processors. Each message data set is accessible to processor A (that is where BDT will execute) and to the processor whose messages it will store. Users submit requests through either processor B or C.

BDT in processor A will write messages destined for users of processor B onto message data set 1. Likewise, BDT in processor A will write messages destined for users of processor C onto message data set 2.

You could allow users to submit requests through processor A. These requests would not be protected by TQI, however. If you wished to protect these requests, you would have to start a TQI address space on processor A and allocate a message data set to this address space.

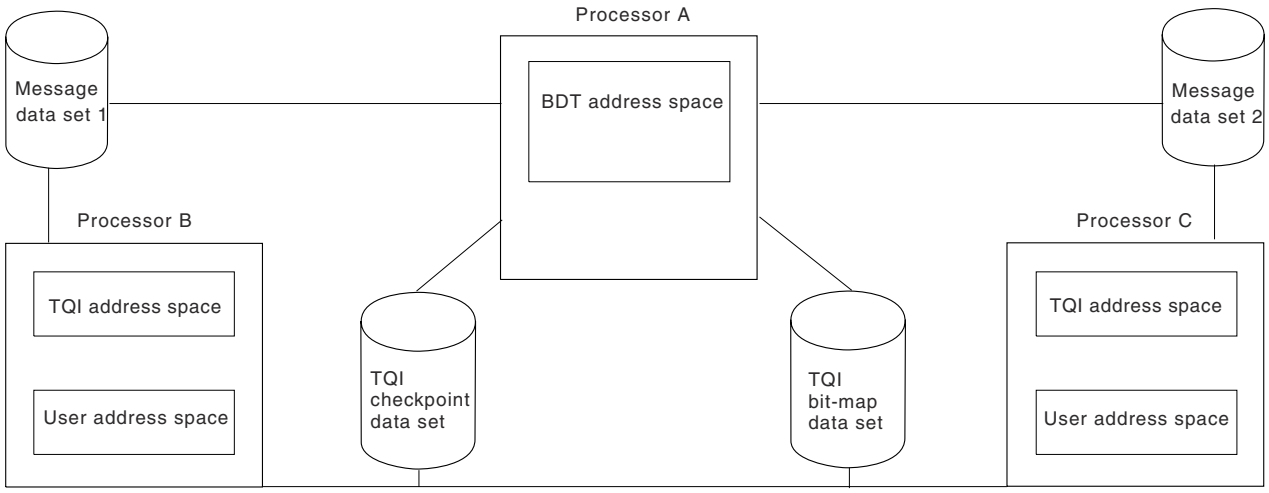
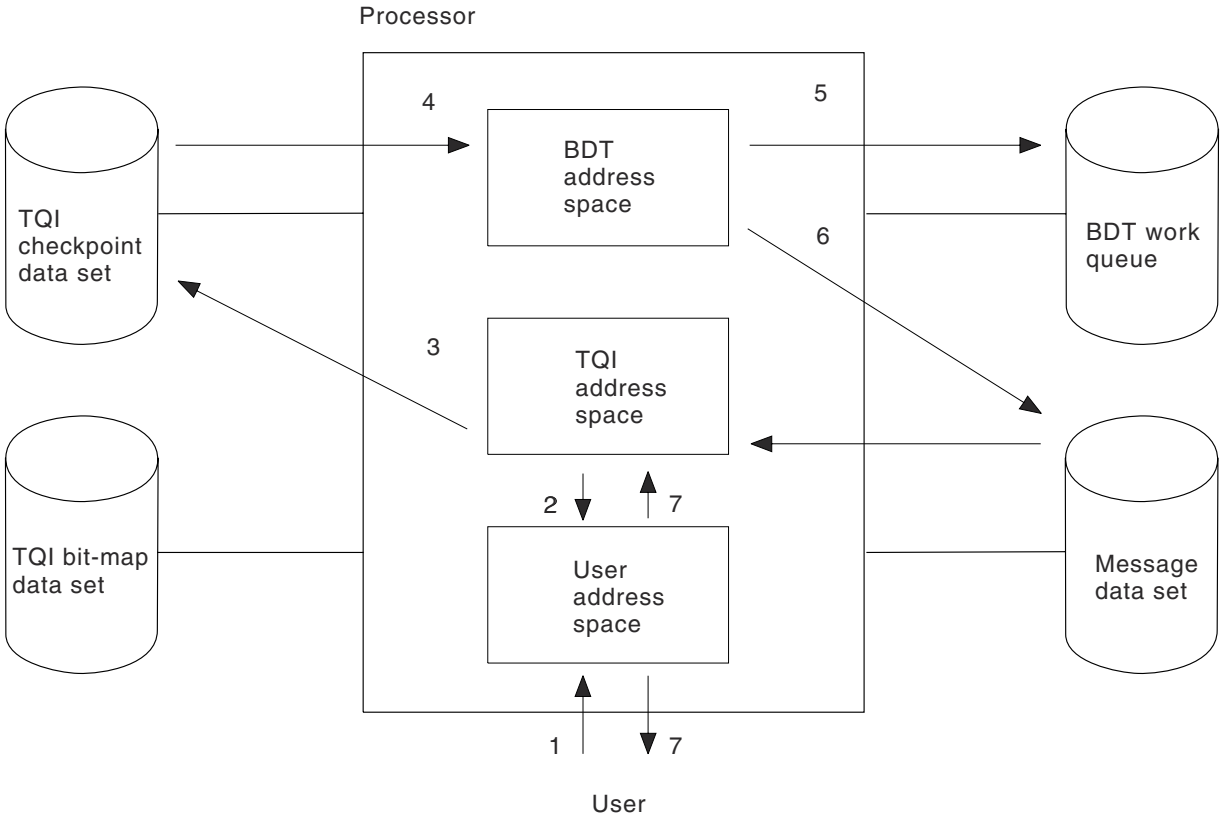


Figure 4. Example of TQI in a three-processor JES complex

Example of TQI flow—TQI and BDT in the same processor

To help you plan your use of TQI you should understand TQI's role in the flow of a command or file-to-file transaction. Figure 5 on page 11 shows the flow of a request submitted by a user in a single-processor configuration, where TQI and BDT are in the same processor.

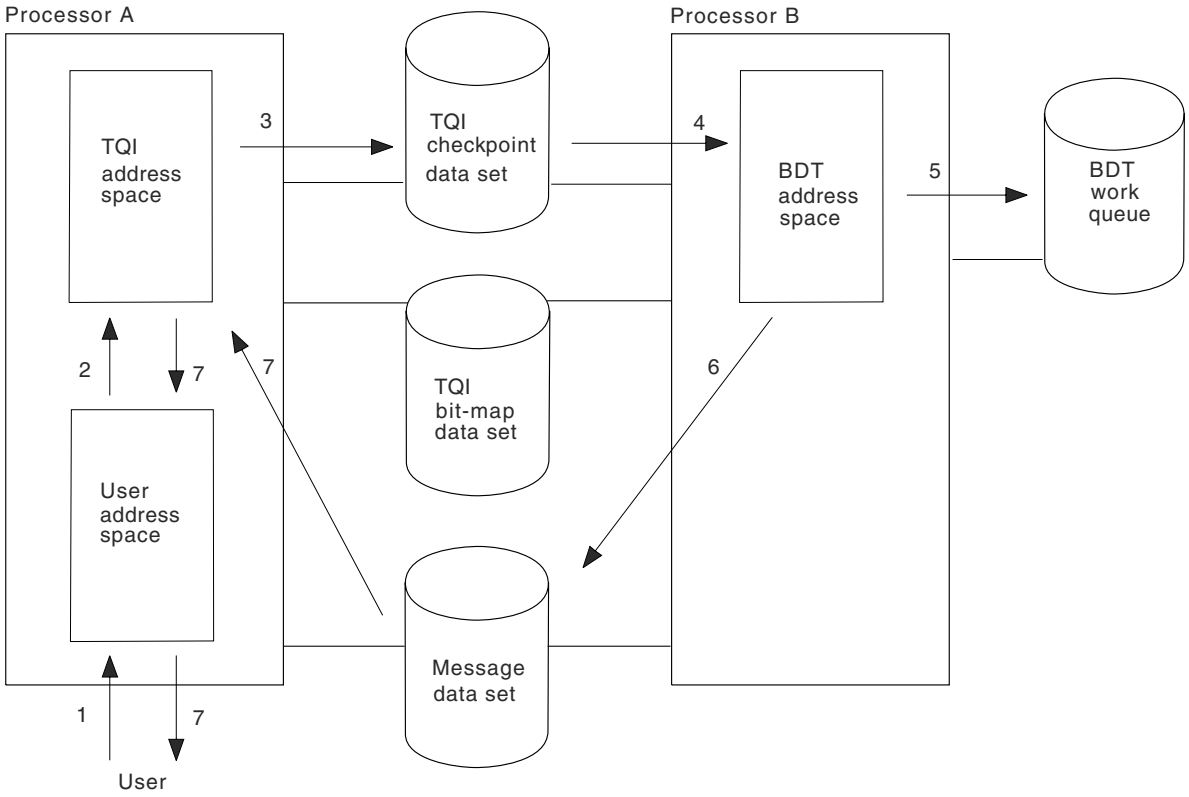


1. A user submits a request (command or file-to-file transaction).
2. The request travels from the user's address space to the TQI address space.
3. TQI writes the request onto the TQI checkpoint data set using space information from the TQI bit-map data set.
4. BDT reads the request from the checkpoint data set.
5. BDT writes the request to the BDT work queue. BDT then notifies TQI that the request was written to the work queue so that TQI can reuse the space on the checkpoint data set that the request had occupied.
6. BDT writes messages for the user onto the message data set. These messages may, for example, report on the progress of a file-to-file transaction or provide information in response to an INQUIRY command.
7. TQI periodically reads messages from the message data set and routes them to the user.

Figure 5. Example of TQI flow with TQI and BDT in the same processor

Example of TQI flow—TQI and BDT in different processors

Figure 6 on page 12 shows the flow of a request submitted by a user in a two-processor complex, where TQI and BDT are in separate processors.



1. A user submits a request (command or file-to-file transaction).
2. The request travels from the user's address space to the TQI address space.
3. TQI writes the request onto the TQI checkpoint data set using space information from the TQI bit-map data set.
4. BDT reads the request from the checkpoint data set.
5. BDT writes the request to the BDT work queue. BDT then notifies TQI that the request was written to the work queue so that TQI can reuse the space on the checkpoint data set that the request had occupied.
6. BDT writes messages for the user onto the message data set. These messages may, for example, report on the progress of a file-to-file transaction or provide information in response to an INQUIRY command.
7. TQI periodically reads messages from the message data set and routes them to the user.

Figure 6. Example of TQI flow with TQI and BDT in different processors

Running without TQI

The use of TQI is optional.

Running without TQI improves system performance. BDT runs faster because there is no message delay. Less resources are needed because a TQI address space is not used. Less I/O takes place because messages are not written to and read from DASD.

However, running without TQI has these disadvantages:

- JES2 users can submit commands and file-to-file transactions only if they are at a processor with a BDT address space. (JES3 users do not have this restriction.)
- BDT must be running when users submit commands and file-to-file transactions. If BDT is down, requests will be lost because they cannot be held on a checkpoint data set until BDT is started or restarted.
- Users can receive messages associated with their commands and file-to-file transactions only if they are at a processor with a BDT address space.

If you choose to run without TQI, the following is necessary:

- You must code TQIEN=N and TQIREQ=N on the IEFSSN*xx* member of SYS1.PARMLIB. (TQIEN=Y and TQIREQ=Y are the defaults. TQIEN and TQIREQ are described on page "Step 1. Define BDT As an MVS Secondary Subsystem—SYS1.PARMLIB Member IEFSSNxx" on page 15.)
- You must not allocate the TQI checkpoint data set, the TQI bit-map data set, or any message data sets in the BDT start procedure. (The BDT start procedure is described on page "Step 1. Write a BDT Start Procedure" on page 61.)
- You must code TQIAUTO=NO on the OPTIONS initialization statement. (TQIAUTO=YES is the default. TQIAUTO is described on page "OPTIONS—Define Operating Characteristics of the BDT Subsystem" on page 51.)

Step 3. Decide whether to have a poly-BDT complex

BDT runs in its own address space. A JES complex may have one or more BDT address spaces. Each processor in the complex may have one or more BDT address spaces. A complex with multiple BDT address spaces is called a poly-BDT complex. Figure 7 on page 13 and Figure 8 on page 13 show examples of poly-BDT complexes.

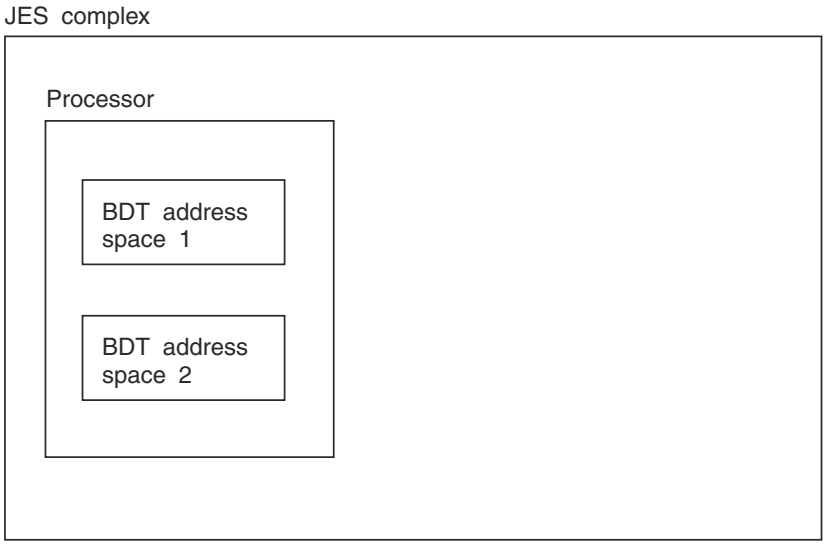


Figure 7. A Poly-BDT complex with BDT address spaces in the same processor

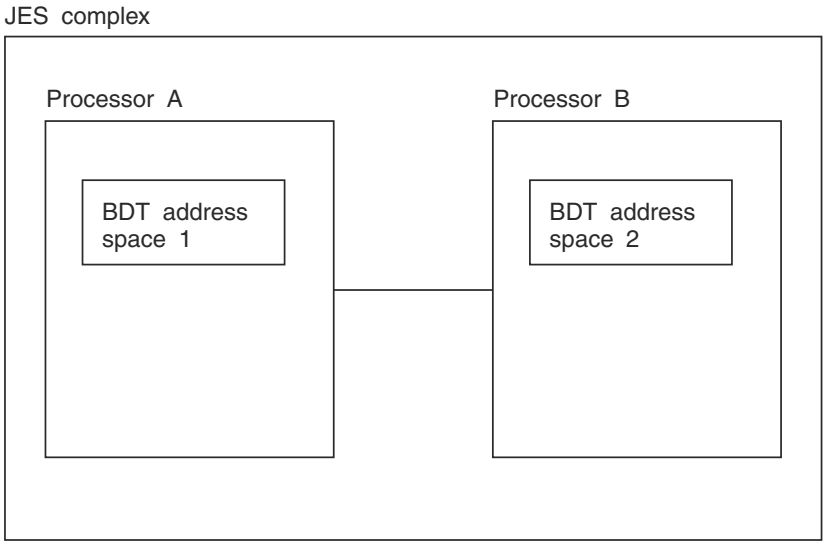


Figure 8. A Poly-BDT complex with BDT address spaces in different processors

A good use for poly-BDT complexes is to separate test work from production work. Figure 9 on page 14 illustrates such a situation. In the figure, complexes A and C are poly-BDT complexes. They each have an address space for daily production work and another address space for testing (such as testing newly written user exit routines) and for training operators and users. By separating testing and training from the production work, a complex reduces the risk of disrupting its production work.

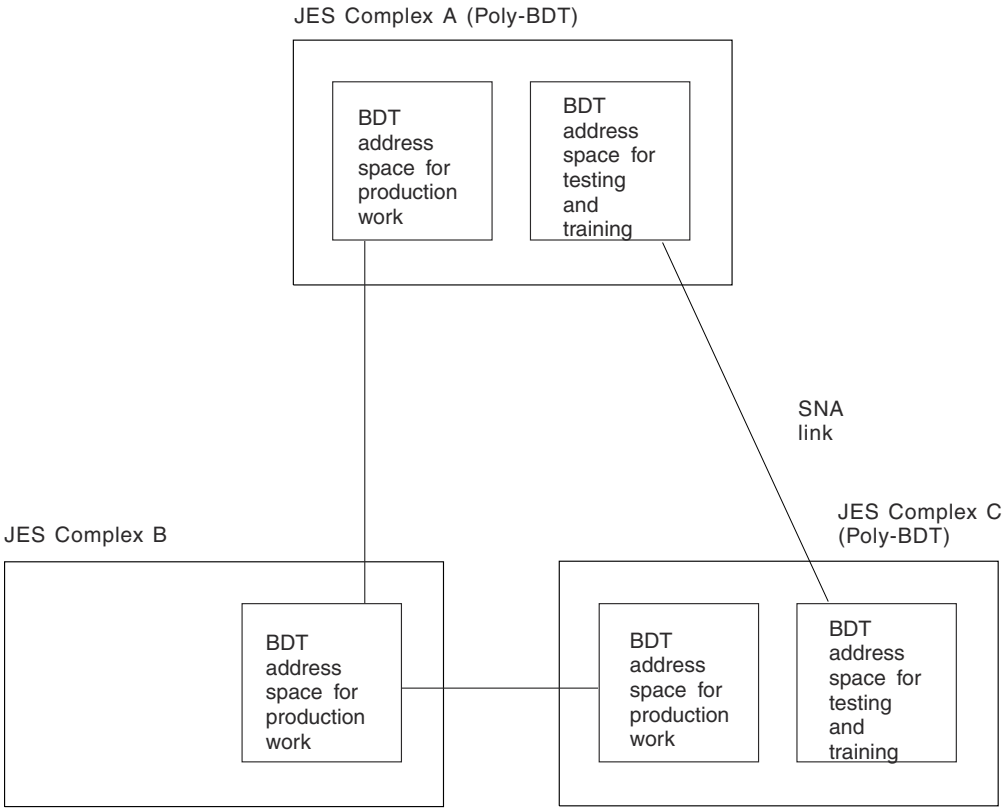


Figure 9. Poly-BDT complexes with separate test and production systems

You must decide how many BDT address spaces to have in a complex and on which processor each should reside. For each BDT address space you must create:

- A separate initialization stream
- A separate BDT start procedure.

If you want to estimate the storage required for each BDT address space, see Appendix B, "Virtual Storage Required for the BDT Address Space," on page 163.

Chapter 3. Defining BDT to MVS

This chapter describes how to:

- Define BDT to MVS as a secondary subsystem by creating an entry in an IEFSSN*xx* member of SYS1.PARMLIB
- Specify system parameters in SYS1.PARMLIB member IEASYS*xx*
- Give authorized program facility (APF) authorization to SYS1.MIGLIB and SYS1.SBDTLIB, the BDT module libraries, by updating SYS1.PARMLIB member IEAAPF*xx* or PROG*xx*
- Define BDT to JES3 if you are a JES3 customer.

Some MVS parameters must be matched with BDT parameters. To get an overall picture of these relationships, refer to Appendix A, “Parameter map,” on page 161.

Step 1. Define BDT As an MVS Secondary Subsystem—SYS1.PARMLIB Member IEFSSNxx

To define BDT as an MVS™ secondary subsystem you must create an entry in an IEFSSN*xx* member of SYS1.PARMLIB.

You must create a member at each processor that will have a BDT address space or a TQI address space. If a processor will have more than one BDT address space you must create one member, with a separate entry for each address space.

The general format of an IEFSSN*xx* entry and the rules for coding and invoking it are described in the *z/OS MVS Initialization and Tuning Guide*. Parameters that apply to BDT are:

BDT Entry in SYS1.PARMLIB Member IEFSSNxx

subsystem-name,BDTSSINI,'*node-name*
[,C=*bdt-char*]
[,D={Y|N}]
[,TQIEN={Y|N}]
[,TQIREQ={Y|N}]'

subsystem-name
is a 1- to 4-character name that you assign to distinguish the BDT secondary subsystem from other MVS secondary subsystems. Later (on “Step 2. Write a TQI Start Procedure” on page 62) you will specify this name as the last one to four characters of the name of the member that contains the TQI start procedure.

BDTSSINI
is the routine that initializes the BDT secondary subsystem.

node-name
identifies the node to which this IEFSSN*xx* entry applies. This parameter must match the name on one of the following SYSID initialization statement parameters:

- The NAME parameter if the subsystem has a file-to-file node, without or in addition to a SNA NJE node
- The NJENAME parameter if the subsystem has only a SNA NJE node

[C=*bdt-char*]
defines the command character for the subsystem defined by this IEFSSN*xx* entry. This character can be prefixed to commands and file-to-file transactions in order to identify them as intended for this subsystem. The default is no command character.

[D={Y|N}]
specifies whether *node-name* is the default BDT node. The default BDT node is the node that executes (1) any BDT commands that do not request a particular node via the SY(*node-name*) prefix and (2) any file-to-file transactions that do not request a particular node via the SYSTEM parameter.

Y
defines the node as the default node.

N
specifies that the node is not the default node.

If you intend to have multiple file-to-file nodes within a JES complex you should code D=Y for one of the nodes. If you fail to do this, commands that do not use the SY(*node-name*) prefix and file-to-file transactions that do not use the SYSTEM parameter will not be accepted from TSO or batch.

[TQIEN={Y|N}]
indicates whether the BDT transaction queuing integrity (TQI) facility is to be automatically enabled after the TQI address space has started.

Y
requests that TQI be automatically enabled.

N
requests that TQI not be automatically enabled. Later the operator may enable TQI by issuing the command MODIFY TQI,E.

[TQIREQ={Y|N}]
indicates whether you require TQI to be enabled before BDT will accept user requests (commands and file-to-file transactions).

Y
requires TQI to be enabled before BDT accepts user requests.

N
allows BDT to accept user requests while TQI is disabled. However, the requests will not be recorded on the TQI checkpoint data set. Therefore, if you choose N, you risk losing user requests. Users must resubmit lost requests; they are not automatically resubmitted by TQI.

Table 1 on page 16 shows the actions taken by BDT and TQI for the various combinations of the TQIREQ and TQIEN parameters. This table assumes that the operator has not altered the TQIREQ and TQIEN settings.

Table 1. TQIREQ and TQIEN Interactions

TQIREQ=	TQIEN=	Action
N	N	TQI does not record user requests on the checkpoint data set. In a JES2 complex, BDT processes only those requests that were submitted through the same processor on which BDT is executing. In a JES3 complex, BDT processes all requests.
Y	N	BDT rejects all user requests.
N	Y	TQI records user requests on the checkpoint data set. BDT processes requests.
Y	Y	TQI records user requests on the checkpoint data set. BDT processes requests.

SYS1.SBDTSAMP contains a sample SYS1.PARMLIB member named BDT$V2SN for defining BDT as a secondary MVS subsystem. It is shown in Figure 10 on page 16.

```
A1,BDTSSINI,'SYSA1,C=/,D=Y,TQIEN=Y,TQIREQ=Y'
```

Figure 10. Sample SYS1.PARMLIB Member IEFSSNxx

Step 2. Specify MVS System Parameters—SYS1.PARMLIB Member IEASYSxx

Several system parameters in SYS1.PARMLIB member IEASYS*xx* are of special importance to BDT:

- To identify the system being initialized you must specify the SYSNAME parameter. The name you specify on SYSNAME you will later specify on the FORMAT control statement used to format message data sets (described on "Step 1. Format the TQI Checkpoint, Bit-Map, and Message Data Sets" on page 31).
- If BDT is to participate in a global resource serialization (GRS) complex you must specify the GRS parameter.

The format of an IEASYSxx entry and the rules for coding it are described in the *z/OS MVS Initialization and Tuning Guide*

Step 3. Authorize SYS1.SBDTLIB

SYS1.SBDTLIB, the BDT module library, requires authorized program facility (APF) authorization. Therefore, you must add the name SYS1.SBDTLIB to SYS1.PARMLIB member IEAAPF*xx* or PROG*xx*.

The format of IEAAPFxx or PROGxx entries and the rules for coding it are described in *z/OS MVS Initialization and Tuning Reference*.

Step 4. Authorize SYS1.MIGLIB

SYS1.MIGLIB, the BDT module library, requires authorized program facility (APF) authorization. Therefore, you must add the name SYS1.MIGLIB to SYS1.PARMLIB member IEAAPF*xx* or PROG*xx*.

The format of IEAAPFxx or PROGxx entries and the rules for coding them are described in the *z/OS MVS Initialization and Tuning Reference*

Step 5. Define BDT to JES3 (JES3 Customers Only)—CONSOLE, SYSID, and NJERMT Statements

If you install BDT in a JES3 complex, you will have to add the following statements to the JES3 initialization stream:

- A CONSOLE statement, which requests JES3 to create a table entry that is required for BDT console support.
- A SYSID statement with the NAME parameter to define the default BDT node to JES3. The default BDT node is the node that executes (1) any BDT commands that do not request a particular node via the SY(*node-name*) prefix and (2) any file-to-file transactions that do not request a particular node via the SYSTEM parameter. The NAME parameter on the JES3 SYSID statement must match the name on one of the following BDT SYSID statement parameters:
  - The NAME parameter if the BDT subsystem has a file-to-file node, without or in addition to a SNA NJE node.
  - The NJENAME parameter if the BDT subsystem has only a SNA NJE node.
- If BDT has the SNA NJE feature, one or more NJERMT statements with:
  - The BDTID parameter to identify the BDT node that receives SNA NJE transactions. The name specified on the BDTID parameter must match the name on the NAME parameter of JES3's SYSID statement.
  - The NAME parameter to identify remote SNA NJE nodes to JES3. The name specified on the NAME parameter must match the name on the N parameter of BDT's BDTNODE statement.

These JES3 statements are explained in the *z/OS JES3 Initialization and Tuning Guide*. To have these statements take effect, the JES3 operator must warm start or cold start JES3.

Chapter 4. Defining BDT to VTAM

This chapter describes how to:

- Define a BDT file-to-file node to VTAM
- Define a BDT SNA NJE node to VTAM
- Define each remote BDT node as a VTAM cross-domain resource
- Define file-to-file session parameters to VTAM
- Define SNA NJE session parameters to VTAM.

Some VTAM parameters must be matched with BDT parameters and other VTAM parameters. To get an overall picture of these relationships, refer to Appendix A, “Parameter map,” on page 161.

Step 1. Define a Node for File-to-File Transfers—APPL Definition Statement

BDT is a VTAM application program. If BDT is to handle file-to-file transfers, you must define BDT to VTAM as a file-to-file node by coding a VTAM APPL definition statement.

Values on the VTAM APPL definition statement that are significant to BDT as a file-to-file node are:

name
is the label on the APPL statement. It must match the name on the APPLID parameter of the BDT SYSID statement, and must also match the names (labels) on the CDRSC statements in the remote nodes that define this application as a cross-domain resource.

AUTH=ACQ
You must code AUTH=ACQ to allow the BDT application to establish sessions using the VTAM SIMLOGON macro.

DLOGMOD=*default logon mode entry name*
Use this operand if you wish to specify a default logon mode name to define the session parameters for sessions in which the BDT node being defined is the secondary end. (Session parameters are discussed in “Step 4. Define File-to-File Session Parameters—Logon Mode Table” on page 22.)

In the absence of this operand, the default logon mode name usually defaults to the first entry in the logon mode table associated with the application (either the VTAM-supplied default logon mode table or the logon mode table specified in the MODETAB operand). When the first entry in the table is not appropriate, you can specify another name as the value in the DLOGMOD operand and include a logmode table entry with this name in the table.

If you use this operand to specify a default logon mode name, you need not specify the L parameter on the remote BDTNODE initialization statements that relate to the BDT node for which the DLOGMOD operand is specified.

EAS=*n*
Specify a value for *n* that is equal to the anticipated number of concurrent sessions that this node will have with all other BDT nodes, plus 10-20 percent for expansion.

MAXPVT=*n*
MAXPVT limits the amount of private area that VTAM attempts to get to satisfy inbound requests. When the limit is reached, if additional inbound data arrives over a session, VTAM terminates the session over which the data was received. It may be useful to restrict the amount of private virtual storage that VTAM allocates in this manner to ensure that there is enough virtual storage to perform other BDT functions. No recommended value can be placed on this operand, as each installation is different. You should consider, however, that it is better to have VTAM take down a BDT session for lack of buffer space than to have the BDT address space terminate with an 80A abend.

Usually, you need not provide a large amount of private buffer area through the MAXPVT operand because BDT allows you to specify the amount of virtual storage to be made available for SNA buffers through SNABUF initialization statements. In this way, you can control the amount of private virtual storage allocated to the SNA buffers.

If you specified sufficient SNA buffers and if sufficient private area exists within the BDT address space to accommodate that definition, you should not need to specify a high MAXPVT value, since BDT should always have enough resources of its own to receive data up to its own pacing limits.

If you provide enough SNA buffers to equal BUFNO + 1 for every outbound VLU, plus BUFNO for every inbound VLU, plus 1 for every communication VLU, you should not need to provide a large amount of private buffer area through the MAXPVT operand.

An exception to the previous rule exists, however, when the receiving side of a BDT connection does not get enough processor cycles to dispatch its VTAM SRB receive exit. This could occur when the BDT dispatching priority is very low, or when a subtask within the BDT address space is abending and a dump is being taken. In these instances, data still arrives from the VTAM interface from the sending BDT node, but VTAM has no BDT SNA buffer in which to place that data. It must, therefore, queue the received data in storage obtained within the BDT address space by means of GETMAIN.

MODETAB=*logon mode table name*
If you use a unique logon mode table for your BDT applications rather than the VTAM default logon mode table, specify its name (*logon mode table name*) on the APPL statement that defines the home BDT node. This unique logon mode table must also include entries to define logon mode names specified on remote BDTNODE initialization statements identifying this BDT node.

PRTCT=*password*
Code this operand only if you code the APPLPSWD parameter on the BDT SYSID statement. Specify the same value.

VPACING=*n*
The value specified on this operand normally governs the flow of inbound data to the application defined by the VTAM APPL statement. However, BDT overrides any value on this operand with 0 to ensure that its sessions are **not** paced through the SNA pacing mechanisms. See the discussion on BDT pacing under the BUFNO parameter of the BDTNODE statement ("BDTNODE—Define Session Characteristics between Home and Remote Nodes" on page 37) to see how BDT implements its own session-level pacing.

SYS1.SBDTSAMP contains a sample SYS1.VTAMLST member named BDT$VTAM that includes an APPL statement for defining BDT as a file-to-file node. It is shown in Figure 11 on page 20.

```
         VBUILD  TYPE=APPL
FTFAPPL1 APPL AUTH=(PASS,NOTSO,NVPACE,ACQ)
NJEAPPL1 APPL AUTH=(PASS,NOTSO,ACQ),VPACING=2
```

Figure 11. Sample SYS1.VTAMLST Member

Step 2. Define a Node for SNA NJE Transfers—APPL Definition Statement

BDT is an VTAM application program. If BDT is to handle SNA NJE transfers, you must define BDT to VTAM as a SNA NJE node by coding an VTAM APPL definition statement.

Values on the VTAM APPL definition statement that are significant to BDT as a SNA NJE node are:

name
is the label on the APPL statement. It must match the name on the NJEAPPL parameter of the BDT SYSID statement, and must also match the names (labels) on the CDRSC statements in the remote nodes that define this application as a cross-domain resource.

AUTH=ACQ
You must code AUTH=ACQ to allow the BDT application to establish sessions using the VTAM SIMLOGON macro.

DLOGMOD=*default logon mode entry name*

Use this operand if you wish to specify a default logon mode name to define the session parameters for sessions in which the BDT node being defined is the secondary end. (Session parameters are discussed in "Step 5. Define SNA NJE Session Parameters—Logon Mode Table" on page 22.)

In the absence of this operand, the default logon mode name usually defaults to the first entry in the logon mode table associated with the application (either the VTAM-supplied default logon mode table or the logon mode table specified in the MODETAB operand). When the first entry in the table is not appropriate, you can specify another name as the value in the DLOGMOD operand and include a logmode table entry with this name in the table.

If you use this operand to specify a default logon mode name, you need not specify the L parameter on the remote BDTNODE initialization statements that relate to the BDT node for which the DLOGMOD operand is specified.

EAS=*n*

Specify a value for *n* that is equal to the anticipated number of concurrent sessions that this node will have with all other BDT nodes, plus 10-20 percent for expansion.

MAXPVT=*n*

MAXPVT limits the amount of private area that VTAM attempts to get to satisfy inbound requests. When the limit is reached, if additional inbound data arrives over a session, VTAM terminates the session over which the data was received. It may be useful to restrict the amount of private virtual storage that VTAM allocates in this manner to ensure that there is enough virtual storage to perform other BDT functions. No recommended value can be placed on this operand, as each installation is different. You should consider, however, that it is better to have VTAM take down a BDT session for lack of buffer space than to have the BDT address space terminate with an 80A abend.

Usually, you need not provide a large amount of private buffer area through the MAXPVT operand because BDT allows you to specify the amount of virtual storage to be made available for SNA buffers through SNABUF initialization statements. In this way, you can control the amount of private virtual storage allocated to the SNA buffers.

If you specified sufficient SNA buffers and if sufficient private area exists within the BDT address space to accommodate that definition, you should not need to specify a high MAXPVT value, since BDT should always have enough resources of its own to receive data up to its own pacing limits.

If you provide enough SNA buffers to equal BUFNO + 1 for every outbound VLU, plus BUFNO for every inbound VLU, plus 1 for every communication VLU, you should not need to provide a large amount of private buffer area through the MAXPVT operand.

An exception to the previous rule exists, however, when the receiving side of a BDT connection does not get enough processor cycles to dispatch its VTAM SRB receive exit. This could occur when the BDT dispatching priority is very low, or when a subtask within the BDT address space is abending and a dump is being taken. In these instances, data still arrives from the VTAM interface from the sending BDT node, but VTAM has no BDT SNA buffer in which to place that data. It must, therefore, queue the received data in storage obtained within the BDT address space by means of GETMAIN.

MODETAB=*logon mode table name*

If you use a unique logon mode table for your BDT applications rather than the VTAM default logon mode table, specify its name (*logon mode table name*) on the APPL statement that defines the home BDT node. This unique logon mode table must also include entries to define logon mode names specified on remote BDTNODE initialization statements identifying this BDT node.

PRTCT=*password*

Code this operand only if you code the NJEAPSWD parameter on the BDT SYSID statement. Specify the same value.

VPACING=*n*

The value for *n* varies from system to system. It is based on the amount of private area available, the number of sessions communicating simultaneously, the size of the SNA buffer pool, the size of the buffers used for each session, and so forth. The recommended initial value is 2. A value of 0 is not recommended because the receiving node has no control over the rate of transmission from the sending node, which could lead to transaction or BDT subsystem failure.

SYS1.SBDTSAMP contains a sample SYS1.VTAMLST member named BDT$VTAM that includes an APPL statement for defining BDT as a SNA NJE node. It is shown in Figure 11 on page 20.

Step 3. Define Remote Nodes as Cross-Domain Resources—CDRSC Definition Statement

You must define each remote node as a VTAM cross-domain resource. You do this by coding a VTAM CDRSC definition statement for each remote node. The name (label) on the CDRSC statement at your node must match the name (label) on the VTAM APPL statement coded for BDT at the remote node. (This name must also match the name on the APPL parameter of the BDTNODE statement that is coded at your node to define the remote node.)

Step 4. Define File-to-File Session Parameters—Logon Mode Table

Whenever a file-to-file session is established between two BDT nodes, the nodes exchange session parameters, which govern the characteristics of the session. You define the session parameters through VTAM logon mode tables. You may define your own logon mode tables and logon mode entries and associate these with the BDT application, or you may accept the VTAM-supplied default logon mode table.

When BDT establishes file-to-file sessions it overrides all session parameters except class of service (COS operand of the MODEENT macro) and cryptography type (ENCR operand of the MODEENT macro). Therefore, if your installation does not need to establish a specific class of service for file-to-file connections, and does not require connection of file-to-file nodes over encrypted sessions, you need not code VTAM logon mode table entries for BDT nor bother with unique MODETAB or DLOGMOD operands on the VTAM APPL definition statements. (MODETAB and DLOGMOD are discussed in “Step 1. Define a Node for File-to-File Transfers—APPL Definition Statement” on page 19.)

The logon mode table values used by BDT for file-to-file sessions are:

```
MODEENT  COMPROT=4020,
         COS=installation-determined,
         ENCR=installation-determined,
         FMPROF=4,
         LOGMODE=LMODFTF,
         PRIPROT=00,
         PSERVIC=000000000000000000000000,
         PSNDPAC=00,
         RUSIZES=8585,
         SECPROT=00,
         SRCVPAC=00,
         SSNDPAC=00,
         TSPROF=7,
         TYPE=0
```

The session parameters that VTAM uses during session establishment always originate from the secondary LU. Therefore, if you want to ensure that a specific set of characteristics are associated with a BDT session, you should ensure that a logon mode table entry defining these characteristics is present at each secondary end of each session that BDT establishes. For file-to-file sessions the local end of the session always becomes the secondary end of the session. Therefore, you should create logon mode table entries that define specific session characteristics at each BDT node that can act as a local node. If a BDT node can act only as a global node, there is no need to define logon mode table entries to VTAM for that BDT application.

Step 5. Define SNA NJE Session Parameters—Logon Mode Table

Whenever a SNA NJE session is established between two BDT nodes, the nodes exchange session parameters, which govern the characteristics of the session. You define the session parameters through VTAM logon mode tables. You may define your own logon mode tables and logon mode entries and associate these with the BDT application, or you may accept the VTAM-supplied default logon mode table.

When BDT establishes SNA NJE sessions it overrides all session parameters except class of service (COS operand of the MODEENT macro), cryptography type (ENCR operand of the MODEENT macro), and pacing

(PSNDPAC, SRCVPAC, and SSNDPAC operands of the MODEENT macro). Therefore, if your installation does not require connection of SNA NJE nodes over encrypted sessions, does not need to establish a specific class of service for SNA NJE connections, and specifies pacing on the VPACING parameter of the APPL definition statement, you need not code VTAM logon mode table entries for BDT nor bother with unique MODETAB or DLOGMOD operands on the VTAM APPL definition statements. (MODETAB and DLOGMOD are discussed in "Step 1. Define a Node for File-to-File Transfers—APPL Definition Statement" on page 19.)

The logon mode table values used by BDT for SNA NJE sessions are:

```
MODEENT  COMPROT=4020,
         COS=installation-determined,
         ENCR=bits 0-1 forced to zero; bits 2-7
             installation-determined,
         FMPROF=3,
         LOGMODE=LMODNJE,
         PRIPROT=72,
         PSERVIC=000000000000000000000000,
         PSNDPAC=installation-determined,
         RUSIZES=0000,
         SECPROT=72,
         SRCVPAC=installation-determined,
         SSNDPAC=installation-determined,
         TSPROF=3,
         TYPE=01
```

The session parameters that VTAM uses during session establishment always originate from the secondary LU. Therefore, if you want to ensure that a specific set of characteristics are associated with a BDT session, you should ensure that a logon mode table entry defining these characteristics is present at each secondary end of each session that BDT establishes.

Chapter 5. Allocating BDT and TQI Data Sets

This chapter describes how to allocate the BDT and TQI data sets needed during BDT operation. The data sets are:

- The BDT initialization stream data set
- The BDT work queue data set
- The system generic master job definition (GMJD) library data set (file-to-file customers only)
- ISPF data sets (file-to-file customers only)
- The TQI checkpoint data set
- The TQI bit-map data set
- Message data sets.

Step 1. Allocate a Data Set for the BDT Initialization Stream

You must allocate a data set to contain the BDT initialization stream (or streams) that is discussed in Chapter 7, "Creating a BDT Initialization Stream," on page 33. You may allocate this data set using ISPF, the TSO ALLOCATE command, or a batch job (using the IEFBR14 program; see Figure 12 on page 25 for sample JCL).

The initialization stream is run when the operator starts BDT. It initializes the BDT address space. You may wish to have several initialization streams. If several streams are available then the operator, by warm starting BDT, can easily change the network configuration or select different options.

Note: A BDT subsystem that has both file-to-file and SNA NJE nodes does not require two initialization streams. Both nodes exist in one BDT address space, so only one initialization stream is needed.

Requirements are:

Name: Any name. In "Step 1. Write a BDT Start Procedure" on page 61you will be directed to specify this name on the BDTIN DD statement of the BDT start procedure.

Size: Customer-determined, depending on the number of initialization statements the data set will contain. The minimum is 16 and there is no maximum. Each initialization statement requires at least one record. (A statement may be continued, requiring more than one record.) Two important factors that affect the size of this data set are the number of comment statements you will use (there is no limit) and the number of remote nodes that your home node will communicate with (one BDTNODE statement is required to specify each remote node, with a limit of 100). The IBM-supplied initialization streams in SYS1.SBDTSAMP have approximately 100 records each.

Organization: The data set may be either sequential or partitioned if it is to contain one initialization stream. The data set must be partitioned if it is to contain more than one initialization stream, and each initialization stream must be a separate member. You can give each member any 3- to 8-character name you want or you can use the form BDTIN*xx*, where *xx* is any one or two alphanumeric characters. The latter method allows the operator a shorthand way of specifying the member name when starting BDT (see "Step 1. Write a BDT Start Procedure" on page 61 for details.)

Record Format: Fixed blocked (FB)

Record Length: 80

Block Size: Any multiple of 80

```
//BDTIN    DD  UNIT=3380,VOL=SER=BDTDRV,DISP=(NEW,CATLG),
//             DSN=SYS1.BDT.INITS,SPACE=(CYL,(5,1,54)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
```

Figure 12. A Sample DD Statement to Allocate a Data Set for the Initialization Stream

Step 2. Allocate a Data Set for the BDT Work Queue

You must allocate a data set to contain the BDT work queue, on which BDT places jobs during BDT operation. You may allocate this data set using ISPF, the TSO ALLOCATE command, or a batch job (using the IEFBR14 program; see Figure 13 on page 26 for sample JCL).

Requirements are:

Name: Any name. In "Step 1. Write a BDT Start Procedure" on page 61 you will be directed to specify this name on the BDSPOOL and CRSPOOL DD statements of the BDT start procedure.

Size: Customer-determined. It is suggested that you initially allocate the equivalent of 2 cylinders on a 3380. For best performance, do not allocate a secondary quantity.

Organization: Any may be specified. BDT changes it to physical sequential (PS).

Record Format: Any may be specified. BDT changes it to fixed (F).

Record Length: Any may be specified. BDT changes it to 0.

Block Size: Any may be specified. BDT changes it to 2048.

```
//BDTSPOOL DD UNIT=3380,VOL=SER=BDTDRV,DISP=(NEW,CATLG),
//            DSN=BDT1.BDTSPOOL,SPACE=(CYL,(2),,CONTIG),
//         DCB=(RECFM=F,LRECL=2048,BLKSIZE=2048,DSORG=PS)
```

Note: During initialization, BDT places a higher than normal demand on BDSPOOL space. Insufficient BDSPOOL space can cause the system to end abnormally (completion codes S060 and BD705). This situation occurs most often during cold starts. If it occurs, increase BDSPOOL space to five contiguous cylinders. The number of cylinders needed may vary with the installation. In any event, the maximum number of cylinders required should not exceed 20 for 3380's or 25 for 3350's.

Figure 13. A Sample DD Statement to Allocate a Data Set for the BDT Work Queue

Step 3. Allocate a System GMJD Library (File-to-File Customers Only)

This step is optional and applies only to systems that have the File-to-File feature of BDT.

A generic master job definition (GMJD) library is a data set that contains precoded file-to-file transaction definitions. There are two types of GMJD libraries: private and system. Users may define private GMJD libraries or they may use a system GMJD library that you create.

If you want to create a system GMJD library you must first allocate a partitioned data set for it. You can allocate it using ISPF, the TSO ALLOCATE command, or a batch job (using the IEFBR14 program; see Figure 14 on page 27 for sample JCL).

Requirements are:

Name: Any name. Later you will specify this name on a DD statement in the BDT start procedure at "Step 1. Write a BDT Start Procedure" on page 61 or on a DYNALLOC statement at "DYNALLOC—Dynamically Allocate BDT Data Sets" on page 49) that is in the initialization stream invoked by the BDT start procedure.

Size: Customer-determined, depending on the total size of all the transaction definitions that will be stored there.

Organization: Partitioned. Each transaction definition must be stored as a member of the data set. The name of the member is also the name of the transaction definition. The member name may be any valid member name except Q or the name of a BDT command.

Record Format: Fixed (F) or fixed blocked (FB)

Record Length: 80

Block Size: Maximum of 32720

Transaction definitions that are stored in a GMJD library may contain passwords that are not encrypted. To prevent unauthorized exposure of these passwords you should RACF®-protect the system GMJD library. Only BDT should be authorized to access the library.

For information about how to use a system GMJD library, see *z/OS BDT File-to-File Transaction Guide*.

```
//GMJDLIB  DD  UNIT=3380,VOL=SER=BDTDRV,DISP=(NEW,CATLG),
//             DSN=SYS1.GMJD,SPACE=(CYL,(10,1,111)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
```

Figure 14. A Sample DD Statement to Allocate a Data Set for a System GMJD Library

Step 4. Allocate ISPF Data Sets (File-to-File Customers Only)

This step is optional and applies only to systems that have the File-to-File feature of BDT.

BDT provides ISPF panels that make it easier for users to submit file-to-file transactions. These panels provide a dialog capability between the user and ISPF. The dialog prompts the user for the information needed to build a BDT transaction. TSO HELP panels and examples guide the user through the task of providing the transaction information. After the user provides the information, ISPF builds a BDT transaction which it then routes to BDT for execution.

ISPF also saves the information that the user provided. The next time the user invokes the ISPF panel, the information that ISPF saved appears on the screen. The user may then submit another transaction using the same information, or the user may change the information before submitting the transaction.

If you want to make the BDT ISPF panels available to your installation you must allocate six data sets and concatenate them with data sets defined in the TSO logon procedure. You may have already allocated the SBDTxxxx data sets during installation of z/OS. The data sets you allocate must have the same characteristics as the data sets with which they are concatenated. See Table 2 on page 27 for the names of the data sets.

Table 2. ISPF Data Sets for BDT

Data Set Name	Data Set Contents	TSO Logon Procedure ddname to Which Data Set Gets Concatenated
SYS1.SBDTHELP	TSO HELP panels	SYSHELP
SYS1.SBDTPN0	BDT ISPF panels	ISPPLIB
SYS1.SBDTMSG	BDT ISPF messages	ISPMLIB
SYS1.SBDTCLI0	BDT ISPF CLISTs	SYSPROC
Any name	Table output library	ISPTABL
Same as ISPTABL data set name	Table input library	ISPTLIB

You have to allocate table input and output library data sets (ddnames ISPTLIB and ISPTABL, respectively) only if you plan to allow users to save their ISPF panels. You must allocate unique data sets for each TSO user. If you do not allocate these data sets, saved transactions will be inaccessible upon reentry to the BDT panels.

ISPF tables stored by users can contain confidential data, such as passwords. To prevent unauthorized access to this data, you should RACF-protect the table output library data sets pointed to by ddname ISPTABL. Access to these data sets should be granted to the users owning them and to the user ID assigned to the BDT started procedure.

You can update the ISPF panels to add the BDT File-to-File feature as an option. The method used to do this depends on which ISPF is installed. Consider incorporating the changes you make in an SMP/E usermod, so that they will not be regressed without warning during the installation of service (PTFs) for ISPF. For more information about constructing SMP/E usermods, see *z/OS SMP/E Reference* or *z/OS SMP/E User's Guide*

ISPF Version 3

If ISPF Version 3 is installed:

Modify an existing ISPF panel (the Utility Selection Panel, ISRUTIL, is recommended) to call CLIST BDTC01, which will display the BDT Selection Menu:

1. Pick an option number that is not used on the panel (Option 15, for example):
2. Add the following to the input panel body:

```
%  15 +BDT - ACCESS BDT (BULK DATA TRANSFER)
```

3. Under the line that reads:

```
&ZSEL = TRANS( TRUNC (&ZCMD,'.'):
```

add the following to the ")PROC" section:

```
15,'CMD(BDTC01)'
```

Other Considerations

If you are running with the Program Cryptographic Facility (PCF) you must define the BDT transaction prefix of “BDT” in the PCF command table before you can send any transactions to BDT through TSO or ISPF.

If BDT is going to be run from an ISPF application ID other than ISR, you must add the parameter NEWAPPL(ISR) to the command that executes the BDT command list in the other application ID.

Step 5. Allocate the TQI Checkpoint Data Set

You must allocate a TQI checkpoint data set if users are to have the commands and file-to-file transactions that they submit checkpointed. There is a limit of one checkpoint data set per JES complex. This data set must be shared by a processor that has a BDT address space and processors that have TQI address spaces. (Refer to “Step 2. Plan the Use of TQI” on page 8 for an introduction to the use of this data set.)

You may allocate this data set using ISPF, the TSO ALLOCATE command, or a batch job (using the IEFBR14 program; see Figure 15 on page 29 for sample JCL).

Requirements are:

Name: Any name. Later you will specify this name on the DATAFILE DD statement of both the BDT start procedure at “Step 1. Write a BDT Start Procedure” on page 61 and the TQI start procedure at “Step 2. Write a TQI Start Procedure” on page 62).
Size: Customer-determined, with a minimum of 1 record and a maximum of 5000 records. One record is required for each checkpointed request (command or file-to-file transaction). To determine the maximum number of requests you want the TQI checkpoint data set to be able to store at one time (and, therefore, the maximum number of records to allocate for the data set), consider the rate at which you expect users to submit requests and the rate at which you expect the BDT address space to be able to read requests from the checkpoint data set. Remember that a request, once written to the checkpoint data set, remains there until BDT writes the request onto the BDT work queue. In the case where TQI is running but BDT is not, the checkpoint data set will continue to fill until BDT is started. For best performance, do not allocate a secondary quantity.
Organization: Any may be specified. BDT changes it to physical sequential (PS).
Record Format: Any may be specified. BDT changes it to fixed (F).
Record Length: Any may be specified. BDT changes it to 3584.
Block Size: Any may be specified. BDT changes it to 3584.

```
//TQIDATA  DD  UNIT=3380,VOL=SER=BDTDRV,DISP=(NEW,CATLG),
//             DSN=BDT1.TQIDATA,SPACE=(CYL,(2),,CONTIG),
//             DCB=(RECFM=F,LRECL=3584,BLKSIZE=3584)
```

Figure 15. A Sample DD Statement to Allocate the TQI Checkpoint Data Set

Step 6. Allocate the TQI Bit-Map Data Set

You must allocate a TQI bit-map data set if you allocate a TQI checkpoint data set. They are used together. You may allocate this data set using ISPF, the TSO ALLOCATE command, or a batch job (using the IEFBR14 program; see Figure 16 on page 29 for sample JCL).

Requirements are:

Name: Any name. Later you will specify this name on the BITMAPS DD statement of both the BDT start procedure at "Step 1. Write a BDT Start Procedure" on page 61 and the TQI start procedure at "Step 2. Write a TQI Start Procedure" on page 62).

Size: Customer-determined, with 2 records required for each 120 requests (commands or file-to-file transactions) that are stored on the TQI checkpoint data set. (Thus, 121 requests would require 4 records.) For best performance, do not allocate a secondary quantity.

Organization: Any may be specified. BDT changes it to physical sequential (PS).

Record Format: Any may be specified. BDT changes it to fixed (F).

Record Length: Any may be specified. BDT changes it to 31.

Block Size: Any may be specified. BDT changes it to 31.

```
//TQIBITS  DD  UNIT=3380,VOL=SER=BDTDRV,DISP=(NEW,CATLG),
//             DSN=BDT1.TQIBITS,SPACE=(TRK,(2,1)),
//             DCB=(RECFM=F,LRECL=31,BLKSIZE=31)
```

Figure 16. A Sample DD Statement to Allocate the TQI Bit-Map Data Set

Step 7. Allocate Message Data Sets

You must allocate one or more message data sets in order for users who submit requests (commands and file-to-file transactions) through TQI to receive messages from BDT. Within a JES complex, each processor that has a TQI address space but not a BDT address space must have a message data set shared by that processor and by a processor that has a BDT address space in order for users at the processor with TQI to receive BDT messages. Thus, a complex should have as many message data sets as it has TQI address spaces. A message data set is not required at a processor that has a BDT address space but not a TQI address space, where users submit requests directly to the BDT address space.

You can allocate message data sets using ISPF, the TSO ALLOCATE command, or a batch job (using the IEFBR14 program; see Figure 17 on page 30 for sample JCL).

Requirements are:

Name: Any name. Later you will specify this name on a BDTM*xxxx* DD statement in the BDT start procedure (In "Step 1. Write a BDT Start Procedure" on page 61) or on a DYNALLOC statement ("DYNALLOC—Dynamically Allocate BDT Data Sets" on page 49) that is in the initialization stream invoked by the BDT start procedure.

Size: Customer-determined, with 6156 bytes required for each message in the data set. The size of the data set at any given time depends on message traffic (the number of messages being written to the data set) and the frequency at which messages are read off the data set (specified as a PARM on the EXEC statement of the TQI start procedure, "Step 2. Write a TQI Start Procedure" on page 62). For best performance, do not allocate a secondary quantity.

Organization: Any may be specified. BDT changes it to physical sequential (PS).

Record Format: Any may be specified. BDT changes it to fixed (F).

Record Length: Any may be specified. BDT changes it to 6156.

Block Size: Any may be specified. BDT changes it to 6156.

```
//TQIMSG   DD  UNIT=3380,VOL=SER=BDTDRV,DISP=(NEW,CATLG),
//             DSN=BDT1.MSG0001,SPACE=(CYL,(2),,CONTIG),
//             DCB=(RECFM=F,LRECL=6156,BLKSIZE=6156)
```

Figure 17. A Sample DD Statement to Allocate a Message Data Set

Chapter 6. Formatting TQI Data Sets

This chapter describes how to format the TQI data sets that you allocated in the preceding chapter. It describes how to format:

- The TQI checkpoint and bit-map data sets
- Message data sets.

Step 1. Format the TQI Checkpoint, Bit-Map, and Message Data Sets

You must format the TQI checkpoint and bit-map data sets that you allocated in "Step 5. Allocate the TQI Checkpoint Data Set" on page 28 and "Step 6. Allocate the TQI Bit-Map Data Set" on page 29, and any message data sets that you allocated in "Step 7. Allocate Message Data Sets" on page 29. You do this by running the BDTTQBCH program from a batch job.

Each time you invoke BDTTQBCH you may use only one of its functions: you may format both a checkpoint data set and a bit-map data set, you may format one message data set, or you may move the contents of a checkpoint data set and a bit-map data set. (Moving the contents of old checkpoint and bit-map data sets to new checkpoint and bit-map data sets becomes necessary if you later discover that the old data sets are too small. For instructions, see Appendix C, "Moving Transactions to a New TQI Checkpoint Data Set," on page 165.) A control statement that you specify as input (SYSIN) to BDTTQBCH selects the appropriate function.

The sample JCL and control statements in SYS1.SBDTSAMP (member name BDT$TQFM) that format the checkpoint, bit-map, and message data sets are shown in Figure 18 on page 31. The sample job contains two steps. The first step uses a BUILD control statement to format the checkpoint and bit-map data sets. The second step uses a FORMAT control statement to format one message data set. Note that the sample assumes that the SBDTLINK library is in the link list and that all data sets are cataloged.

```
//BDTTQBCH  JOB  CLASS=A
//TQIBATCH  EXEC PGM=BDTTQBCH
//BITMAPS   DD   DISP=SHR,DSN=BDT1.TQIBITS
//DATAFILE  DD   DISP=SHR,DSN=BDT1.TQIDATA
//SYSUDUMP  DD   SYSOUT=A
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
BUILD,SYSID=SYSA1
//*
//TQIMSG    EXEC PGM=BDTTQBCH
//MESSAGE   DD   DISP=SHR,DSN=BDT1.MSG0001
//SYSUDUMP  DD   SYSOUT=A
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
FORMAT,SYSID=SYSA1,SYSNAME=XXXXXXXX
//*
```

Figure 18. Sample Job to Format the TQI Checkpoint, Bit-Map, and Message Data Sets

//TQIBATCH EXEC
identifies BDTTQBCH as the program to be executed.

//BITMAPS DD
defines the TQI bit-map data set. The data set name must be the one you specified when you allocated the data set in "Step 6. Allocate the TQI Bit-Map Data Set" on page 29.

//DATAFILE DD
defines the TQI checkpoint data set. The data set name must be the one you specified when you allocated the data set in "Step 5. Allocate the TQI Checkpoint Data Set" on page 28.

//SYSUDUMP DD
defines the data set where a formatted storage dump is to be written in the event BDTTQBCH abnormally terminates. You may replace the SYSUDUMP DD statement with a SYSMDUMP DD statement or a SYSABEND DD statement, depending on the type of dump you want.

//SYSPRINT DD
defines the data set where BDTTQBCH is to write its messages.

//SYSIN DD
defines the input to BDTTQBCH.

BUILD
is the control statement that formats the checkpoint and bit-map data sets. It can have two parameters:

,SYSID=*node-name*
identifies the BDT node that will read the TQI checkpoint data set. *node-name* must match the name on one of the following SYSID initialization statement parameters:

- The NAME parameter if the BDT subsystem has a file-to-file node, without or in addition to a SNA NJE node
- The NJENAME parameter if the BDT subsystem has only a SNA NJE node.

This parameter is required. The example uses SYSA1.

,RECORDS=*nnnn*
defines the number of records that are to be created on the TQI checkpoint data set. The number of records corresponds to the maximum number of requests (commands and file-to-file transactions) that the data set can store. *nnnn* must be a decimal number from 1 to 5000. If you omit this parameter, as many records are created (up to a maximum of 5000) as the allocated space allows. The example omits this parameter.

//TQIMSG EXEC
identifies BDTTQBCH as the program to be executed.

//MESSAGE DD
defines the message data set that is to be formatted. The data set name must be the one you specified when you allocated the data set on "Step 7. Allocate Message Data Sets" on page 29.

FORMAT
is the control statement that formats the message data set. It requires two parameters:

,SYSID=*node-name*
identifies the BDT node that will write messages to the message data set. *node-name* must match the node name on the SYSID parameter of the BUILD control statement. This parameter is required. The example uses SYSA1.

,SYSNAME=*system-name*
identifies the MVS system that contains the BDT node identified by *node-name*. *system-name* must match the SYSNAME parameter that has been specified for BDT in an IEASYS*xx* member of SYS1.PARMLIB (as described on "Step 2. Specify MVS System Parameters—SYS1.PARMLIB Member IEASYSxx" on page 17). This parameter is required. The example uses XXXXXXXX.

Certain types of errors, such as programming errors in user-written exit routines, can prevent BDT from reading the checkpoint data set. Permanent data errors on the checkpoint data set may also cause this problem. The only way to correct the problem is to reformat the checkpoint and bit-map data sets. After reformatting the data sets you must restart TQI and BDT. If you determine that an exit routine caused the problem, you should either fix the routine or disable it before doing the restart. Transactions that were on the checkpoint data set when the problem occurred are lost. If a user still wishes to execute one of these transactions, the user must resubmit the transaction.

Chapter 7. Creating a BDT Initialization Stream

This chapter describes the initialization statements (which make up an initialization stream) that you must put into the data set you allocated in “Step 1. Allocate a Data Set for the BDT Initialization Stream” on page 25. The initialization stream is run when the operator starts BDT. It initializes the BDT address space. This chapter describes:

- The number of initialization streams you should have.
- The IBM-supplied initialization streams in SYS1.SBDTSAMP.
- The rules for coding initialization statements.
- The format of each initialization statement. The statements are presented in alphabetic order (for easy reference), not in the order in which they must appear in the initialization stream. (“Rules for coding initialization statements” on page 35 shows the required order.)
- Initialization statement parameters that the operator can override.

Many initialization statement parameters must be matched with MVS parameters, VTAM parameters, and other BDT parameters. To get an overall picture of these relationships, refer to Appendix A, “Parameter map,” on page 161.

How Many Initialization Streams Should You Have?

You will probably want to have more than one initialization stream. If several streams are available then the operator, by warm starting BDT, can easily change the network configuration or select different options.

If your BDT system has both file-to-file and SNA NJE nodes you do not need separate initialization streams for each. Both nodes exist in one BDT address space, so only one initialization stream is needed.

The IBM-Supplied Initialization Streams

You can code your own initialization statements from scratch or you can modify the ones in SYS1.SBDTSAMP. Regardless of the method you choose, follow the rules and formats presented in this chapter.

The sample initialization streams in SYS1.SBDTSAMP are shown in Figure 19 on page 34, Figure 20 on page 34, and Figure 21 on page 35.

```
CELLPOOL,ID=CSRB,CNUM=256,SCNUM=256,MAXET=1,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=DCQE,CNUM=10,SCNUM=5,MAXET=5,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=FCT,CNUM=51,SCNUM=30,MAXET=8,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=ICMB,CNUM=15,SCNUM=15,MAXET=5,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=IFC,CNUM=48,SCNUM=30,MAXET=6,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=JCTB,CNUM=112,SCNUM=112,MAXET=8,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=JML,CNUM=6,SCNUM=6,MAXET=15,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=OCMB,CNUM=80,SCNUM=64,MAXET=4,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=SAVE,CNUM=128,SCNUM=128,MAXET=5,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=SICA,CNUM=16,PGRLSE=YES
CELLPOOL,ID=TQCP,CNUM=60,SCNUM=120,MAXET=4,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=TQE,CNUM=113,SCNUM=113,MAXET=2,AUTODEL=YES,PGRLSE=YES
*********************************************************************
DYNALLOC,DDN=GMJDLIB,DSN=SYS1.GMJD,UNIT=3380,VOLSER=BDTDRV
*********************************************************************
OPTIONS,JOBNO=(1,100,0),SYSLOG=(JES3,PRINT),                        X
SYSMSG=YES,WANTDUMP=ASK,JES3=YES
*********************************************************************
ENDRBAM
*********************************************************************
SYSID,NJENAME=SYSA1,APPLID=FTFAPPL1
*********************************************************************
BDTNODE,N=SYSA1,LU=4,BUFSZ=1024,BUFNO=2
BDTNODE,N=SYSA2,LU=8,BUFSZ=256,BUFNO=10,CKPT=100,CS=(NJEDUP,REPDUP),  X
APPL=FTFAPPL2
BDTNODE,N=SYSA3,LU=8,BUFSZ=1024,BUFNO=10,CS=NJEDUP,T=LOCAL,          X
APPL=FTFAPPL3
*********************************************************************
SNABUF,SIZE=1024,PRI=80,SEC=(25,3),AUTODEL=YES
SNABUF,SIZE=256,PRI=100,SEC=(100,24),AUTODEL=YES
*********************************************************************
ENDINIT
```

Figure 19. Sample BDT File-to-File Initialization Stream

```
CELLPOOL,ID=CSRB,CNUM=256,SCNUM=256,MAXET=1,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=DCQE,CNUM=10,SCNUM=5,MAXET=5,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=FCT,CNUM=51,SCNUM=30,MAXET=8,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=ICMB,CNUM=15,SCNUM=15,MAXET=5,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=IFC,CNUM=48,SCNUM=30,MAXET=6,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=IFCN,CNUM=113,SCNUM=113,MAXET=3,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=JCTB,CNUM=112,SCNUM=112,MAXET=8,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=JML,CNUM=6,SCNUM=6,MAXET=15,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=OCMB,CNUM=80,SCNUM=64,MAXET=4,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=SAVE,CNUM=128,SCNUM=128,MAXET=5,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=SICA,CNUM=16,PGRLSE=YES
CELLPOOL,ID=TQCP,CNUM=60,SCNUM=120,MAXET=4,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=TQE,CNUM=113,SCNUM=113,MAXET=2,AUTODEL=YES,PGRLSE=YES
*********************************************************************
OPTIONS,JOBNO=(1,100,0),SYSLOG=(JES3,PRINT),                        X
SYSMSG=YES,WANTDUMP=ASK,JES3=YES
*********************************************************************
ENDRBAM
*********************************************************************
SYSID,NJENAME=SYSA1N,NJEAPPL=NJEAPPL1
*********************************************************************
BDTNODE,N=SYSA2N,LU=9,BUFSZ=1024,BUFNO=8,TYPE=NJE,APPL=NJEAPPL2
BDTNODE,N=SYSA3N,LU=9,BUFSZ=1024,BUFNO=8,TYPE=NJE,APPL=NJEAPPL3
*********************************************************************
SNABUF,SIZE=1024,PRI=80,SEC=(25,3),AUTODEL=YES
*********************************************************************
ENDINIT
```

Not all comment lines are shown.

Figure 20. Sample BDT SNA NJE Initialization Stream (for a JES3 System)

```
CELLPOOL,ID=CSRB,CNUM=256,SCNUM=256,MAXET=1,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=DCQE,CNUM=10,SCNUM=5,MAXET=5,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=FCT,CNUM=51,SCNUM=30,MAXET=8,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=ICMB,CNUM=15,SCNUM=15,MAXET=5,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=IFC,CNUM=48,SCNUM=30,MAXET=6,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=IFCN,CNUM=113,SCNUM=113,MAXET=3,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=JCTB,CNUM=112,SCNUM=112,MAXET=8,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=JML,CNUM=6,SCNUM=6,MAXET=15,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=OCMB,CNUM=80,SCNUM=64,MAXET=4,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=SAVE,CNUM=128,SCNUM=128,MAXET=5,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=SICA,CNUM=16,PGRLSE=YES
CELLPOOL,ID=TQCP,CNUM=60,SCNUM=120,MAXET=4,AUTODEL=YES,PGRLSE=YES
CELLPOOL,ID=TQE,CNUM=113,SCNUM=113,MAXET=2,AUTODEL=YES,PGRLSE=YES
*********************************************************************
DYNALLOC,DDN=GMJDLIB,DSN=SYS1.GMJD,UNIT=3380,VOLSER=BDTDRV
*********************************************************************
OPTIONS,JOBNO=(1,100,0),SYSLOG=(JES3,PRINT),                        X
SYSMSG=YES,WANTDUMP=ASK,JES3=YES
*********************************************************************
ENDRBAM
*********************************************************************
SYSID,NAME=SYSA1,APPLID=FTFAPPL1,NJENAME=SYSA1N,NJEAPPL=NJEAPPL1
*********************************************************************
BDTNODE,N=SYSA1,LU=4,BUFSZ=1024,BUFNO=2
BDTNODE,N=SYSA2,LU=8,BUFSZ=256,BUFNO=10,CKPT=100,CS=(NJEDUP,REPDUP),  X
APPL=FTFAPPL2
BDTNODE,N=SYSA2N,LU=9,BUFSZ=492,BUFNO=8,TYPE=NJE,APPL=NJEAPPL2
BDTNODE,N=SYSA3,LU=8,BUFSZ=1024,BUFNO=10,CS=NJEDUP,T=LOCAL,          X
APPL=FTFAPPL3
BDTNODE,N=SYSA3N,LU=9,BUFSZ=492,BUFNO=8,TYPE=NJE,APPL=NJEAPPL3
*********************************************************************
SNABUF,SIZE=1024,PRI=80,SEC=(25,3),AUTODEL=YES
SNABUF,SIZE=492,PRI=80,SEC=(80,5),AUTODEL=YES
SNABUF,SIZE=256,PRI=100,SEC=(100,24),AUTODEL=YES
*********************************************************************
ENDINIT
```

Not all comment lines are shown.

Figure 21. Sample BDT File-to-File and SNA NJE Initialization Stream (for a JES3 System)

Rules for coding initialization statements

The initialization statements must appear in the following order:

Order	Statement	Optional or Required	Where Statement Is Described
Anywhere	Comment	Optional	“Place Comments in the Initialization Stream” on page 36
First	CELLPOOL	Required	“CELLPOOL—Allocate Cell Pools” on page 44
Second	DYNALLOC	Optional	“DYNALLOC—Dynamically Allocate BDT Data Sets” on page 49
Third	OPTIONS	Optional	“OPTIONS—Define Operating Characteristics of the BDT Subsystem” on page 51
Fourth	ENDRBAM	Required	“ENDRBAM—Mark the End of Definitions So Far” on page 50
Next, in any order	BDTNODE	Required	“BDTNODE—Define Characteristics of a Home File-to-File Node” on page 36, “BDTNODE—Define Session Characteristics between Home and Remote Nodes” on page 37
	SNABUF	Required	“SNABUF—Define Data Buffers” on page 56
	SYSID	Required	“SYSID—Name the Home Node” on page 58
Last	ENDINIT	Required	“ENDINIT—End the Initialization Stream” on page 50

Note: In this chapter the initialization statements are presented in alphabetic order (for easy reference), not in the order required in the initialization stream.

The first blank character that BDT encounters on a statement marks the end of the parameters on that statement. BDT treats the rest of the statement as a comment.

All initialization statements may be continued. To continue a statement:

1. Break the statement after the comma following a complete parameter.
2. Code a nonblank character in column 72.
3. Start the next statement (the continuation statement) in column 1.

BDT logs each initialization statement that it reads. If BDT detects an error in a statement it also logs the error message following the statement. BDT logs this information in the data set defined by the BDTOUT DD statement in the BDT start procedure. When an error occurs, BDT attempts to continue with initialization.

Place Comments in the Initialization Stream

You may use this statement to place comments in the initialization stream.

How Many Allowed: 0-unlimited.

How Many Required: None.

Placement: Anywhere in the initialization stream.

Comment Statement

**comment*

*****
identifies this statement as a comment statement.

comment
is the text of the comment. The maximum length is 70 characters.

BDTNODE—Define Characteristics of a Home File-to-File Node

This form of the BDTNODE statement specifies information about buffers and VLUs for intra-node file transfers within a home file-to-file node. In intra-node communication, only the home node (the node you are at) takes part. That is, the data set being copied and the copy are directly accessible to the home node.

This form of the BDTNODE statement is not used for SNA NJE nodes.

How Many Allowed: 1-100 BDTNODE statements (home and remote combined) per initialization stream.

How Many Required: One BDTNODE statement is required to define the home node if it is a file-to-file node.

Placement: Between the ENDRBAM and ENDINIT statements, before or after the SNABUF and SYSID statements.

BDTNODE Statement for a Home File-to-File Node

Defaults:

BDTNODE,N=*home-node-name*

[,BUFNO=*num-buffs*]
2

[,BUFSZ=*buff-size*]
1024

[,LU=*num-vlus*]
7

N=*home-node-name*
is the name of the home file-to-file node. Operators and users will use this name to identify the home node in commands and file-to-file transactions. BDT will use the name in messages to identify the home node.

Specified As: One to eight alphanumeric characters, the first of which must be alphabetic.

Default: None.

Related Parameters: *home-node-name* must match the name on the NAME parameter of the SYSID statement. For JES3 installations, *home-node-name* and the name on the NAME parameter of the JES3 NJERMT statement must specify **different** names.

[BUFNO=*num-buffs*]
defines the maximum number of data buffers that are available for intra-node communication at a home file-to-file node.

Specified As: A decimal number from 1 to 255.

Default: 2.

[BUFSZ=*buff-size*]
defines the data buffer size (in bytes) that is to be used for intra-node communication at a home file-to-file node.

Specified As: A number of bytes. It may be a decimal number from 0 to 4096.

Default: 1024.

Related Parameters: *buff-size* must not exceed the largest SIZE parameter specified on a SNABUF statement. If it does, initialization will terminate.

[LU=*num-vlus*]
defines the maximum number of virtual logical units (VLUs) that are available for intra-node communication at a home file-to-file node.

Specified As: A decimal number from 1 to 255. The first VLU (the communication VLU) transfers control information and the remaining VLUs (up to 254) transfer data sets.

Default: 7.

BDTNODE—Define Session Characteristics between Home and Remote Nodes

This form of the BDTNODE statement defines characteristics of the communication sessions between the home node (the node you are at) and remote nodes (nodes at other BDT subsystems). The BDTNODE statement can define a pacing rate, a checkpoint interval, data compression type, the use of automatic session startup and restart, the number of VLUs available, and other characteristics.

This form of the BDTNODE statement may be used for file-to-file or SNA NJE nodes.

How Many Allowed: 1-100 BDTNODE statements (home and remote combined) per initialization stream.

How Many Required: One or more BDTNODE statements are required to define sessions with remote nodes (one BDTNODE statement for each remote file-to-file or SNA NJE node).

Placement: Between the ENDRBAM and ENDINIT statements, before or after the SNABUF and SYSID statements.

BDTNODE Statement for Sessions with Remote Nodes

Defaults:

BDTNODE,N=*remote-node-name*,APPL=*lu-name*

[,A={YES|NO}]
NO

[,ASR={YES|NO|*restart- limit*}]
YES

[,BUFNO=*num-buffs*]
2

[,BUFSZ=*buff-size*]
1024

[,CKPT=*K-bytes*]
40

[,CS={NJEDUP|REPDUP|(NJEDUP,REPDUP)}
No compress

[,L=*logmode*]
Standard

[,LU=({*num-vlus*}, {*fence-from*},{*fence-to*})]

[,PIN=*receive-password*]
8 blanks

[,POUT=*send-password*]
8 blanks

[,T=LOCAL]
Remote local

[,TYPE={FTF|NJE}]
FTF

N=*remote-node-name*
specifies a remote file-to-file or SNA NJE node. You must name each remote node that will establish a session with the home node. Users and operators at the home node will use this name to direct commands, file-to-file transactions, and SNA NJE jobs to the remote node. You must use a different name for each remote node. No remote node may have the same name as your home node.

Specified As: One to eight alphanumeric characters, the first of which must be alphabetic.

Default: None.

Related Parameters: For SNA NJE installations, be sure that *remote-node-name* is the same as the name specified on the NAME parameter of the corresponding JES3 NJERMT statement. Do not use the names ALL, NJE, or FTF.

APPL=*lu-name*
is the name that VTAM uses for the remote node being defined by this BDTNODE statement.

Specified As: One to eight alphanumeric characters.

Default: None.

Related Parameters: *lu-name* must match the name (label) of the VTAM CDRSC definition statement at the home node and the name (label) of the VTAM APPL definition statement at the remote node.

[A={YES|NO}]
defines whether BDT at the home node is to automatically start a session with BDT at the remote node named on this statement.

YES
instructs BDT to automatically start a session with the remote node each time the operator at the home node starts BDT and activates the BDT SNA manager.

NO
makes the operator responsible for starting the session.

Default: NO.

[ASR={YES|NO|*restart-limit*}]
defines whether BDT automatically tries to restart a failed session with the remote node. BDT can try to restart failing sessions with the remote node with the exception of sessions that failed because they were canceled by the BDT CANCEL command, and sessions that could not start because of a negotiable bind disagreement between the home node and the remote node.

YES
requests that BDT try automatic session restart an unlimited number of times.

NO
requests that BDT not attempt automatic session restart.

restart-limit
requests that BDT try automatic restart up to *restart-limit* number of times before giving up. *restart-limit* may be a decimal number from 1 to 32767.

Default: YES.

Related Parameters: The ASRTIME parameter of the OPTIONS statement specifies the time delay between restart attempts. This time delay applies to all sessions for which ASR is specified in the BDTNODE statement.

[BUFNO=*num-buffs*]
defines the pacing rate for communication that takes place between the remote node and the home node.

Each time the home node sends data to a remote node, it does so by sending the data in a data buffer. The home node retains a copy of this buffer until the remote node acknowledges receipt of the buffer. If a node has several VLUs actively sending data, each VLU could have sent one or more buffers for which an acknowledgment has not been received. Each of these buffers will occupy virtual storage until the acknowledgment is received.

The pacing rate defines the maximum number of data buffers that each home node VLU can send to the remote node before the remote node must acknowledge receipt of the buffer. If a VLU has sent the maximum number of buffers without receiving an acknowledgment, the VLU stops sending data buffers. The VLU resumes sending data buffers only after it receives the proper acknowledgment from the remote node.

The pacing rate that you define at your node may be different from the pacing rate defined at the remote node. If this happens, BDT uses the slower pacing rate (the smaller *num-buffs* value).

When selecting a pacing rate, you must consider the amount of virtual storage required for the buffers versus BDT's performance. Initially, you should start with a pacing rate of 2 (the default). You should increase this value only if doing so improves the data throughput of BDT.

Specified As: A decimal number from 1 to 255.

Default: 2.

Related Parameters: If *num-buffs* exceeds the number of buffers specified on the SNABUF statement, BDT abend BD615 could occur. For example, if you defined BUFNO=30 on the BDTNODE statement and PRI=10,SEC=0 on the SNABUF statement, and if the 10 primary buffers were sent with no pacing response, the buffer pool could become exhausted.

File-to-File Consideration: For inbound VLUs, BDT receives and queues up to the value of *num-buffs* because that is the maximum that can be sent prior to an acknowledgment. For outbound VLUs, BDT allows up to the value of *num-buffs*+1 SNA buffers to be queued for transmission for each VLU, and transmits up to the value of *num-buffs* until an acknowledgment is received.

As an example of the requirements for outbound SNA buffers, consider a connection that has five VLUs defined—one control VLU and four data transfer VLUs (two outbound and two inbound)—and BUFNO=3. Then six is the maximum number of SNA buffers that can be sent by each side of the session before receiving acknowledgment from the other side.

A slow receiver generally results in an accumulation of SNA buffers at the receiving side of the session, because the SNA buffers used at the sending side are released when the VTAM SEND

completes, not when the BDT acknowledgment is received from the receiving VLU. The SNA buffers at the receiving side remain allocated and in use until the receiving side actually processes the data. Therefore, for the previous example, and with a slow receiver, up to 15 SNA buffers may be in use at any given time. This includes up to eight outbound buffers (even though a maximum of six can be sent) as well as six inbound buffers and a control VLU buffer.

SNA NJE Consideration: For SNA NJE nodes, the communication VLU sends and receives data, and half of the data transfer VLUs send data and half receive data. So the total number of SNA buffers this session can use is (((*num-buffs*/2)+1) x (*num-vlus*+1)), where *num-vlus* is the number of VLUs specified on the LU parameter of this BDTNODE statement.

[BUFSZ=*buff-size*]
defines the data buffer size (in bytes) that is to be used for communication between the remote node and the home node. These buffers are allocated from the pool of buffers that you define on one or more SNABUF statements.

Specified As: A number of bytes. For file-to-file nodes, *buff-size* may be a decimal number from 0 to 4096. For SNA NJE nodes, *buff-size* may be a decimal number from 300 to 4096.

Default: 1024.

Related Parameters: *buff-size* must not exceed the largest SIZE parameter specified on a SNABUF statement. If it does, initialization will terminate.

When selecting a *buff-size* value, you should match the value to the speed of the communication line. For slow speed lines, use a *buff-size* value of 512 bytes or less. Increase the size only for high speed lines or if a larger size improves data throughput. A *buff-size* value that is too large may delay communication traffic and increase the use of virtual storage.

The amount of available space within a SNA buffer where BDT can store data for file transfers is reduced by one byte for each negotiated VLU present on the connection. This space is used to transfer BDT acknowledgments. The negotiated VLU corresponds to the smaller of the two values specified in the LU parameter of the BDTNODE statements for the two nodes in session.

In addition, a small amount of space within the SNA buffer is used for control information on each transfer, and is not available for transferring user file data. The reserved space varies with the logical record size being transferred.

In general, to calculate the amount of space available for transfer of file data within a SNA buffer, reduce the value of *buff-size* by 3, and subtract one byte for each negotiated VLU and two bytes for each logical record contained within the buffer. The previous formula is an average, and the control information may vary somewhat at the start of a file transfer.

For instance, if a given connection has 30 VLUs defined at both nodes, *buff-size* is 256 bytes, and the length of the logical records is 80, the amount of file transfer data that could fit within each SNA buffer is 217, or 256 – (3+(1x30)) = 223 – (3 logical records x 2) = 217.

Note the following for cross-domain and channel-to-channel transmission:

- **For cross-domain transmission:** The value obtained by adding *buff-size* plus the length of the VTAM path information unit (PIU) header *must be less than or equal to* the value specified on the MAXDATA parameter of the VTAM PCCU macro for the network control program (NCP).

  The computation of (MAXBFRU x UNITSZ), which are the two parameters on the NCP HOST macro, must be equal to or greater than the largest PIU that can flow in the network. Therefore, an installation must ensure that, when specifying these values for the NCP, the product of the two is at least as large as the largest *buff-size* + space for the TH and RH portions of the PIU, for any connection that may go through that NCP.

  The MAXBFRU and UNITSZ operands on the NCP HOST macro are used in other VTAM parameter calculations (such as IOBUF size and XPANPT for the IOBUF pool). Consult VTAM publications for additional information.

- **For channel-to-channel transmission:** *buff-size* must be less than or equal to the value obtained by multiplying the MAXBUFRU parameter on the GROUP statement for a CTC link group times the size of the VTAM I/O buffer.

[CKPT=*K-bytes*] (file-to-file nodes only)
Periodically, BDT takes a checkpoint of sequential data set transfers. If the transfer fails and has to be restarted, the checkpoint information that BDT records enables BDT to restart the transfer from the last checkpoint. If the checkpoint were not taken, BDT would have to retransfer the entire data set. *K-bytes* defines how many 1K-byte blocks should be transferred before BDT checkpoints sequential data set transfers. (For a partitioned data set, BDT automatically takes a checkpoint after copying each member. You need not take any action.)

Specified As: A decimal number from 8 to 32767, indicating 1K-byte blocks of uncompressed data before the transfer.

Default: 40.

When you select a checkpoint frequency, try to match the frequency to the speed of the communication link. For slower speed links, you should take checkpoints more frequently than you take them for faster links. Each time BDT takes a checkpoint, communication takes place between the local node and the global node. Therefore, frequent checkpoints on fast links could create excessive communication traffic and adversely affect the performance of the link.

[CS={NJEDUP|REPDUP|(NJEDUP,REPDU P)}] (file-to-file nodes only)
defines the type of data compression, if any, that the home node will allow for file-to-file sessions with the remote node specified on this BDTNODE statement. Data compression removes duplicate strings of characters from data sets to be transferred. The character strings are removed before the data sets are transferred.

NJEDUP
requests compression of 3 to 63 duplicate characters (2 to 63 if the characters are blanks).

REPDUP
requests compression of 3 to 127 duplicate characters.

(NJEDUP,REPDUP)
requests compression of 3 to 127 duplicate characters (2 to 127 if the characters are blanks).

Default: If the CS parameter is not specified, compression does not take place.

The actual compression that takes place is determined by the CS parameter specified at this node, the CS parameter specified at the partner node, and the CSOPT transaction parameter specified by the user, according to the following table.

Table 3. CSOPT Transaction Parameters

CS Parameter Specified at the Home Node	CS Parameter Specified at the Remote Node	CSOPT Transaction Parameter Specified by the User	Compression Used
(NJEDUP,REPDUP)	(NJEDUP,REPDUP)	NJEDUP	NJEDUP
(NJEDUP,REPDUP)	(NJEDUP,REPDUP)	REPDUP	REPDUP
(NJEDUP,REPDUP)	(NJEDUP,REPDUP)	None	None
NJEDUP	(NJEDUP,REPDUP)	NJEDUP	NJEDUP
NJEDUP	(NJEDUP,REPDUP)	REPDUP	None
NJEDUP	(NJEDUP,REPDUP)	None	None
REPDUP	(NJEDUP,REPDUP)	NJEDUP	None
REPDUP	(NJEDUP,REPDUP)	REPDUP	REPDUP

Table 3. CSOPT Transaction Parameters (continued)			
CS Parameter Specified at the Home Node	**CS Parameter Specified at the Remote Node**	**CSOPT Transaction Parameter Specified by the User**	**Compression Used**
REPDUP	(NJEDUP,REPDUP)	None	None
None	(NJEDUP,REPDUP)	NJEDUP	None
None	(NJEDUP,REPDUP)	REPDUP	None
None	(NJEDUP,REPDUP)	None	None
REPDUP	NJEDUP	NJEDUP	None
REPDUP	NJEDUP	REPDUP	None
REPDUP	NJEDUP	None	None

[L=*logmode*]
identifies the entry in the logmode table that VTAM is to use when the remote node named on this BDTNODE statement and the home node try to establish a session.

Specified As: One to eight alphanumeric characters. The first character must be alphabetic.

Default: If this parameter is not specified, VTAM uses the first entry from the standard IBM-supplied logmode table.

Related Parameters: Before you code *logmode*, ensure that the named entry has been defined in the ACF/VTAM logmode table associated with the VTAM APPL definition statement at the node defined by this BDTNODE statement.

[LU=({*num-vlus*},{*fence-from*},{*fence- to*})]
defines the number of virtual logical units (VLUs) that are available for communication between the remote node and the home node. The number of VLUs determine how many concurrent data transfers can take place between the remote node and the home node. This parameter also defines, for file-to-file transfers, how these VLUs are to be fenced.

num-vlus
is the total number of VLUs that are to be available. The first VLU (the communication VLU) always transfers control information. The remaining VLUs transfer data.

For File-to-File Nodes, Specified As: A decimal number from 1 to 255. The first VLU transfers control information and the remaining VLUs (up to 254) transfer data sets.

Default for File-to-File Nodes: 7.

For SNA NJE Nodes, Specified As: 5, 9, 13, 17, 21, 25, or 29. The first VLU transfers control information and the remaining VLUs (up to 28, divided into groups of four) transfer jobs and SYSOUT. Each group has one VLU for handling jobs received, one for jobs sent, one for SYSOUT received, and one for SYSOUT sent. See Table 4 on page 42 for a summary of SNA NJE VLU allocation.

Default for SNA NJE Nodes: 5.

Table 4. Number of SNA NJE VLUs for Each num-vlus Value					
num-vlus Value	**Number of Communication VLUs**	**Number of VLUs for Jobs Received**	**Number of VLUs for Jobs Sent**	**Number of VLUs for SYSOUT Received**	**Number of VLUs for SYSOUT Sent**
5	1	1	1	1	1

Table 4. Number of SNA NJE VLUs for Each num-vlus Value (continued)

num-vlus Value	Number of Communication VLUs	Number of VLUs for Jobs Received	Number of VLUs for Jobs Sent	Number of VLUs for SYSOUT Received	Number of VLUs for SYSOUT Sent
9	1	2	2	2	2
13	1	3	3	3	3
17	1	4	4	4	4
21	1	5	5	5	5
25	1	6	6	6	6
29	1	7	7	7	7

When you allocate VLUs, keep in mind that:

- It may take only a few active VLUs to saturate the capacity of a communication line.
- If the total number of VLUs active across all sessions is too large, the storage capacity of BDT private virtual storage may be exceeded.
- Transferring a data set that has a large block size uses more private virtual storage than does transferring a data set of a smaller block size.

***fence-from* (file-to-file nodes only)**
***fence-to* (file-to-file nodes only)**

Many file-to-file transactions, each wishing to transfer data in the same direction, could enter the system. These transactions could monopolize the VLUs for some time. During this time, other transactions wishing to transfer data in the other direction would be unable to do so. Whether this situation would ever occur at your node depends on the number and characteristics of the transactions that users submit. You can prevent this situation from happening, however, by fencing VLUs.

Fencing is a way of reserving some VLUs for transactions that will transfer data in a specific direction. You can fence VLUs for transactions that will send data to another node; you can also fence VLUs for transactions that will receive data from another node.

For example, assume that you define nine VLUs and that you fence four in the “from” direction and two in the “to” direction. BDT may pair the four “from” VLUs only with transactions that are to receive data from the remote node and the two “to” VLUs only with transactions that will send data to the remote node. BDT may pair the remaining VLUs with any transaction.

You may fence VLUs only for a remote node that is local with respect to the home node. The remote node is local if this BDTNODE statement **does not** contain the T=LOCAL parameter.

fence-from defines the number of VLUs that may be used only to receive data from the remote node. *fence-to* defines the number of VLUs that may be used only to send data to the remote node.

Specified As: Each parameter may be a decimal number from 0 to 254.

Default: 0.

The sum of the values of *fence-from* and *fence-to* must be at least 1 less than the value of *num-vlus*. This is because BDT uses one VLU to pass control information and this VLU is unavailable for data transfers.

[PIN=*receive-password*]
[POUT=*send-password*]

For each remote node you may define a pair of passwords. When the home node and the remote node try to establish a session, BDT checks the passwords. If either the home node or the remote node has failed to provide the correct password, BDT will not allow the session.

receive-password is the password that the home node expects to receive from the remote node. *send-password* is the password that the remote node expects to receive from the home node.

Specified As: Each parameter may be one to eight alphanumeric characters.

Default: A character string of eight blanks (hex 4040404040404040).

[T=LOCAL] (file-to-file nodes only)
defines the global-local relationship between the home file-to-file node and a remote file-to-file node.

The global-local relationship determines which of the two nodes schedules and manages work that takes place between the nodes. Before you decide which node should be global, consider that:

- All transactions are queued and scheduled at the global node.
- The operator at the global node generally needs to be more experienced than the operator at the local node.

A given node may be global in its relationship with one remote node and local in its relationship with a different remote node.

In a JES3 complex, the BDT global-local relationship has no relationship to the JES3 global-local relationship.

See "Step 1. Plan Global and Local Relationships (File-to-File Customers Only)" on page 5 for more information.

Specified As: T=LOCAL if the home file-to-file node (the node at which this BDTNODE statement will execute) is local with respect to the remote file-to-file node (the node defined by this BDTNODE statement), code T=LOCAL. If the home file-to-file node is global with respect to the remote file-to-file node, omit this parameter.

Default: If you omit this parameter from a file-to-file initialization stream, the home node is defined as global with respect to this remote node. You should omit this parameter from a SNA NJE initialization stream, although if you specify it will be ignored.

Related Parameters: T=LOCAL and fencing (specified by the LU parameter) cannot be specified on the same BDTNODE statement; fencing is not allowed from a local node to a global node.

You must ensure that you and the system programmer at the remote node define the same global-local relationship. If you define your node as the global node, the system programmer at the remote node must also define your node as the global node; if you define your node as the local node, the other system programmer must do likewise. If you and the other system programmer fail to define the same global-local relationship, the home node and the remote node will be unable to establish a session.

[TYPE={FTF|NJE}]
specifies the type of data transfers that may occur between the home node and the remote node specified on this BDTNODE statement.

FTF
specifies that the home and remote BDT nodes may send and receive only data sets. This parameter corresponds to the BDT File-to-File feature.

NJE
specifies that the home and remote BDT nodes may send and receive only SNA NJE jobs and SYSOUT. This parameter corresponds to the BDT SNA NJE feature for JES3 installations.

Default: FTF.

CELLPOOL—Allocate Cell Pools

A cell (sometimes called a storage cell) is another name for the main storage that is defined by a CELLPOOL statement. A cell pool is a group of cells that are related by the fact that BDT uses them for the same purpose.

Figure 22 on page 45 shows the structure of a cell pool. It shows that:

- Cells are divided into groups called extents. There are primary extents and secondary extents.
- Each cell pool must have one primary extent and may optionally have up to 99 secondary extents.
- Each extent may contain up to 4096 cells.
- All cells in a pool are the same size. The sizes are listed in Table 5 on page 48.
- The minimum number of cells in a pool is one. The maximum number is 409600 (4096 cells in each extent x 100 extents).

How Many Allowed: 12 or 13 per initialization stream, as described subsequently.

How Many Required: A BDT subsystem with only a file-to-file node requires 12 different cell pools (all except the one with ID=IFCN). A subsystem with only a SNA NJE node requires 12 different cell pools (all except the one with ID=DCQE). A subsystem with both a file-to-file node and a SNA NJE node, requires all 13 cell pools unless:

- The file-to-file feature is *not* present in BDTLIB. In this case, the cell pool with ID=DCQE is not required.
- The SNA NJE feature is *not* present in BDTLIB. In this case, the cell pool with ID=IFCN is not required.

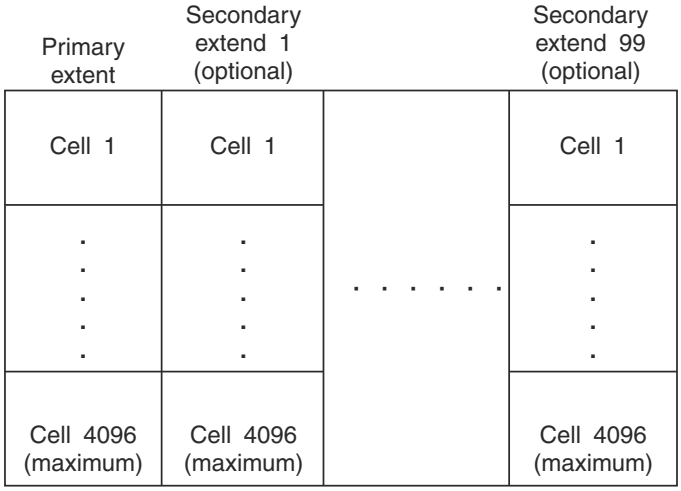


Figure 22. Cell pool

Each cell pool has a unique ID and must be defined on a separate CELLPOOL statement. Table 5 on page 48 lists the IDs of the 13 possible cell pools.

Placement: Except for comment statements, which may appear anywhere if used, CELLPOOL statements must be first in an initialization stream. You may code the CELLPOOL statements in any order.

After you have defined your cell pools and have a test system running, you can use the INQUIRY,C command to monitor BDT's use of cell pools. (For a description of this command see *z/OS BDT Commands* This command provides a report about the amount of storage allocated and the amount of storage used in each cell pool. With this information, you will be able to decide whether you allocated too little or too much cell pool storage. You will then be able to adjust the cell pool allocations accordingly.

CELLPOOL Statement

Defaults:

CELLPOOL,ID=*cell-id*,CNUM=*num-cells*

[,AUTODEL={YES|NO}]
NO

[,MAXET=*max-extents*]
0

[,PGRLSE={YES|NO}]
YES

[,SCNUM=*num-cells*]
4

[,SPAN={YES|NO}]
NO

ID=*cell-id*
is the identifier of the cell.

Specified As: One of the IDs listed in Table 5 on page 48.

Default: None.

CNUM=*num-cells* (pertains to primary extent)
is the minimum number of cells that BDT is to allocate for the primary extent. There is one primary extent per cell pool and it exists for the life of the cell pool. The primary extent may contain up to 4096 cells.

Specified As: A decimal number from 4 to 4096. See Table 5 on page 48 for recommended initial values.

Default: None.

BDT allocates cell pool storage in full-page multiples. This means that BDT may allocate more cells than you request on the CNUM parameter.

For example, if you code the following statement:

```
CELLPOOL,ID=CSRB,CNUM=30
```

360 bytes of storage are needed to satisfy your cell pool request, calculated as follows:

```
      30 (number of cells requested)
    x 12 (size of one CSRB cell)
   _____
     360 bytes
```

BDT allocates one full page (4096 bytes) for the CSRB cell pool. Thus, the cell pool will actually contain 341 cells:

```
     4096 (size of a page)
    ____________________________  = 341 cells
      12  (size of a CSRB cell)
```

[AUTODEL={YES|NO}] (pertains to secondary extents)
determines whether BDT deletes empty secondary extents.

YES
instructs BDT to delete empty secondary extents.

NO
instructs BDT to keep empty secondary extents.

Default: NO.

Some cell pools will become very large as the demand for their storage increases. When the demand for their storage decreases these same cell pools will become much smaller. To conserve virtual storage, specify AUTODEL=YES for these cell pools. The affected cell pools are: IFC, JCTB (if you request resident JCTs on the JOBNO parameter of the OPTIONS statement), OCMB, SAVE, and TQCP.

Note: The SICA cell pool does not use secondary extents. If you specify AUTODEL for the SICA cell pool, the specification will be ignored.

[MAXET=*max-extents*] (pertains to secondary extents)
is the maximum number of secondary storage extents that BDT may allocate. BDT allocates a secondary extent only after the primary extent or previously allocated secondary extents are full. Like the primary extent, secondary extents may contain up to 4096 cells.

Specified As: A decimal number from 0 to 99. See Table 5 on page 48 for recommended initial values.

Default: 0.

Related Parameters: If you specify MAXET you must also specify SCNUM.

Note: The SICA cell pool does not use secondary extents. If you specify MAXET for the SICA cell pool, the specification will be ignored.

[PGRLSE={YES|NO}] (pertains to primary and secondary extents)
determines whether BDT releases empty real storage pages in the primary or secondary extents. Page frames are removed from real storage as soon as BDT finishes with them. But BDT will not release a page if any cell in that page is in use. Freeing unused cell pool storage may reduce paging I/O operations for unused cell pool pages.

YES
instructs BDT to release empty pages.

NO
instructs BDT to keep empty pages.

Default: YES.

Related Parameters: If you code SPAN=YES, you must code PGRLSE=NO.

Another way to free cell pool storage is with the AUTODEL parameter.

[SCNUM=*num-cells*] (pertains to secondary extents)
is the number of cells that may be allocated to each secondary extent.

Specified As: A decimal number from 4 to 4096. See Table 5 on page 48 for recommended initial values.

Default: 4.

Note: The SICA cell pool does not use secondary extents. If you specify SCNUM for the SICA cell pool, the specification will be ignored.

[SPAN={YES|NO}] (pertains to primary and secondary extents)
determines whether cells can span page boundaries.

In deciding whether to allow cells to span page boundaries, consider the amount of paging activity versus the amount of wasted virtual storage. If you allow cells to span page boundaries, you increase the probability that a cell will be split across two pages when a page is written to the page data set. When a cell has been split because of paging, MVS may have to read both pages into real storage before BDT can use the cell.

When you do not allow cells to cross page boundaries, you increase the probability that virtual storage will be wasted. For example, a cell that is 1280 bytes long would not fit into a page that had 1000 bytes unused. If this example were to occur, BDT would have to allocate the cell to the next full page and 1000 bytes would be wasted.

YES
allows cells to span page boundaries.

NO
does not allow cells to span page boundaries.

Default: NO.

Related Parameters: If you specify PGRLSE=YES (the default) you must specify SPAN=NO (the default).

Table 5. Information for Coding CELLPOOL Statements

Cell Pool ID (and Name)	Subpool from Which Allocated	Size of Each Cell (in Bytes)	Number of Cells per Page	Recommended Initial Value			Allocation Considerations
				CNUM	SCNUM	MAXET	
CSRB (common services request block)	13	16	256	256	256	1	This is a low activity cell pool. You should seldom need to allocate more than 341 cells (one page).
DCQE (dynamic application program checkpoint queue element)	16	816	5	10	5	5	If you change the checkpoint frequency or change the speed of the communication lines, you should review the output of the INQUIRY, C command to determine if you need to change the size of this cell pool.
FCT (function control table pool)	15	1364	3	51	30	8	This cell pool must contain 20 cells for BDT plus 1 cell for each active transaction.
ICMB (input console message buffer)	12	260	15	15	15	5	This is usually a low activity pool. Use may increase during session restart.
IFC (interfunction communication)	19	816	5	48	30	6	This cell pool is volatile. That is, for short periods of time there may be a demand for a large number of cells. This type of demand may occur, for example, during a session restart. After the restart completes, the demand for cells will decrease.
IFCN (interfunction communication for NJE)	21	36	113	113	113	3	This cell pool is for SNA NJE nodes only, not file-to-file nodes.
JCTB (job control table buffer)	14	256	16	112	112	8	If you do not request resident JCTs, allocate one cell for each active transaction. If you do request resident JCTs, allocate one cell for each active transaction and one cell for each inactive (on the BDT work queue) transaction.
JML (job message log buffer)	18	676	6	6	6	15	This is a low activity cell pool. You should seldom need to allocate more than six cells (one page).

Table 5. Information for Coding CELLPOOL Statements (continued)

Cell Pool ID (and Name)	Subpool from Which Allocated	Size of Each Cell (in Bytes)	Number of Cells per Page	Recommended Initial Value			Allocation Considerations
				CNUM	SCNUM	MAXET	
OCMB (output console message buffer)	11	256	16	80	64	4	The amount of space you allocate will depend on the number of INQUIRY commands issued and on the amount of output generated.
SAVE (save area pool)	10	128	32	128	128	5	This cell pool must contain four to five cells per active transaction.
SICA (scheduler interface control area)	230	256	16	16	Not used	Not used	This cell pool is used for relatively short periods of time. It does not use secondary extents.
TQCP (transaction queuing cell pool)	20	68	60	60	120	4	This cell pool is volatile. That is, for short periods of time there may be a demand for a large number of cells. This type of demand may occur, for example, during a session restart. After the restart completes, the demand for cells will decrease.
TQE (timer queue element buffer pool)	17	36	113	113	113	2	This is a low activity cell pool. You should seldom need to allocate more than 113 cells (one page).

DYNALLOC—Dynamically Allocate BDT Data Sets

Some of the data sets that must be allocated during BDT startup can be specified using DYNALLOC statements in the initialization stream rather than using DD statements in the BDT start procedure.

You may use DYNALLOC statements to allocate the data sets identified by the following ddnames:

- BDTM*xxxx*—message data sets
- GMJDLIB—the system GMJD library data set.

You may **not** use DYNALLOC statements to allocate the data sets identified by the following ddnames:

- BDTIN—the data set that contains the BDT initialization stream
- BDTOUT—the data set to which BDT writes initialization statements and initialization messages
- BDSPOOL and CRSPOOL—the BDT work queue.

To allocate a data set using DYNALLOC, include a DYNALLOC statement for it in the BDT initialization stream and do not include a DD statement for it in the BDT start procedure.

Do not confuse the use of DYNALLOC to allocate data sets to BDT during startup with the allocations of new data sets to the system described in Chapter 5, “Allocating BDT and TQI Data Sets,” on page 25.

How Many Allowed: 0-288 per initialization stream.

How Many Required: None.

Placement: After the CELLPOOL statements and before the OPTIONS statement. If OPTIONS is not used, DYNALLOC must be placed before the ENDRBAM statement.

DYNALLOC Statement

```
DYNALLOC,DDN=dd-name,DSN=ds-name,UNIT=device-type
    [,VOLSER=serial-number]
```

DDN=*dd-name*
is the ddname that you want to assign to this allocation.

Specified As: *dd-name* must follow MVS conventions for ddnames.

Default: None.

DSN=*ds-name*
is the name of the data set that is to be allocated.

Specified As: *ds-name* must follow MVS conventions for data set names.

Default: None. You must specify this parameter if you are dynamically allocating a data set. However, if you are dynamically allocating a unit record device rather than a data set, this parameter does not apply.

UNIT=*device-type*
is the generic device type on which the data set resides.

Specified As: A device type such as 3380.

Default: None. You must specify this parameter if you are dynamically allocating an uncataloged data set or a unit record device. However, if you are dynamically allocating a cataloged data set, this parameter is optional.

[VOLSER=*serial-number*]
is the serial number of the volume on which the data set resides.

Specified As: *serial-number* must follow MVS naming conventions for volume serial numbers.

Default: None. If you are dynamically allocating a cataloged data set, this parameter is optional. If you are dynamically allocating an uncataloged data set, this parameter is optional but using it ensures that the correct volume is mounted. If you are dynamically allocating a unit record device, this parameter does not apply.

ENDINIT—End the Initialization Stream

The ENDINIT statement marks the end of the initialization stream.

How Many Allowed: One per initialization stream.

How Many Required: One.

Placement: Last in the initialization stream.

ENDINIT Statement

```
ENDINIT
```

ENDRBAM—Mark the End of Definitions So Far

The ENDRBAM statement marks the end of the CELLPOOL, DYNALLOC, and OPTIONS part of the initialization stream.

How Many Allowed: One per initialization stream.

How Many Required: One.

Placement: After the OPTIONS statement. If OPTIONS is not used, after the DYNALLOC statement. If DYNALLOC is not used, after the CELLPOOL statements.

ENDRBAM Statement

ENDRBAM

OPTIONS—Define Operating Characteristics of the BDT Subsystem

You can use the OPTIONS statement to specify operating characteristics of the home BDT subsystem.

How Many Allowed: One per initialization stream.

How Many Required: None; default values are taken for all options if this statement is not specified.

Placement: After the DYNALLOC statement, or after the CELLPOOL statements if DYNALLOC is not used, and before the ENDRBAM statement.

OPTIONS Statement

Defaults:

OPTIONS

[,ACCINT=*frequency*]
10

[,ASRTIME=*time-delay*]
60

[,AUTORS={YES|NO}]
NO

[,BDTRACF={YES|NO}]
YES

[,DUMP={BDT|PRDMP}]
BDT

[,GDGLOCS={YES|NO}]
NO

[,JES3={YES|NO}]
NO

[,JOBNO=({*low-job-no*},{*high-job-no*}, {*resident-jcts*})]
1,1000,0

[,JOBRETPD=*retention-period*]
0

[,LOGCLASS=*print-class*]
A

[,LOGLIMIT=*line-limit*]
999999

[,LOGPAGE=*number-of-lines*]
60

[,MAXTRAN=*concurrent-transfers*]
64

[,MSGPROP={YES|NO}]
YES

[,SYSLOG=({JES3|PRINT|WTO})]
PRINT

[,SYSMSG={YES|NO}]
NO

[,TQIAUTO={YES|NO}]
YES

[,TQITIME=*read-frequency*]
30

[,URSCNT=*threshold*]
0

[,WANTDUMP={ASK|YES|NO}]
YES

[ACCINT=*frequency*]
When users submit file-to-file transactions they specify the maximum amount of processor time (from 1 second to 24 hours) each transaction may use. *frequency* defines how often BDT is to calculate the amount of processor time used.

Specified As: A decimal number from 10 to 900, indicating hundredths of a second.

Default: 10 (which specifies a frequency of .10 of a second).

Related Parameters: This parameter is related to the TIME parameter that a user may code on a file-to-file transaction. The TIME parameter defines the total amount of processor time a transaction may use.

[ASRTIME=*time-delay*]
specifies the amount of time that BDT is to delay between the time a session ends abnormally and the time BDT tries to restart the session. This time delay allows BDT to do any cleanup work that must be done when a session ends abnormally.

Specified As: A decimal number from 0 to 99999, indicating time in seconds.

Default: 60 (which specifies 60 seconds).

[AUTORS={YES|NO}]
determines whether BDT is to restart itself automatically after it abnormally terminates.

YES
instructs BDT to restart automatically.

NO
instructs BDT not to restart automatically. The operator will have to restart BDT.

[BDTRACF={YES|NO}]
determines whether userids and passwords are verified for users submitting BDT transactions.

YES
instructs BDT to request verification of userids and passwords.

NO
bypasses verification of userids and passwords. All users can access BDT authorized data sets. RACF security verification still occurs at the BDT task level.

Default: YES.

[DUMP={BDT|PRDMP}]
defines the type of storage dump BDT is to produce if BDT terminates abnormally and you specified WANTDUMP=ASK or WANTDUMP=YES (the default).

BDT
requests a formatted dump. BDT writes this dump to the data set defined by the BDTABEND DD statement in the BDT start procedure.

PRDMP
requests an unformatted dump. BDT writes this dump to the SYS1.DUMP data set.

Default: BDT.

Related Parameters: In order for the DUMP parameter to take effect you must specify WANTDUMP=ASK or WANTDUMP=YES (the default) on this OPTIONS statement.

The dump data sets may contain sensitive information such as unencrypted RACF passwords. To ensure that such information is not exposed to unauthorized individuals, you should provide security for these data sets. You may do this by RACF-protecting the data sets.

[GDGLOCS={YES|NO}]
determines whether the system will resolve a GDG relative generation number based on the most recent catalog information or on the catalog information that was available the first time the GDG was referenced by BDT.

YES
requests the system use a LOCATE to determine the relative generation mumber based on the most recent catalog information.

NO
requests the systme to determine the relative generation number based on the catalog information that was available the first time the GDG was referenced by BDT.

Default: NO.

If the default is taken of NO is explicitly specified, then the first time the TO section of a transaction specifies a generation data group (GDG) relative data set number, BDT creates a new data set. Thereafter, each time the same relative data set number is specified, BDT overlays the data set. BDT continues to do this until the BDT address space is restarted. Therefore, you should not usually refer to a GDG by means of a relative data set number.

[JES3={YES|NO}]
determines whether BDT is to establish a communication path between itself and the JES3 global processor. This communication path must be established before BDT can receive commands or transactions that have been submitted through JES3.

YES
instructs BDT to establish a communication path between itself and the JES3 global processor.

NO
instructs BDT not to establish a communication path between itself and the JES3 global processor.

Default: NO on JES3 systems. On JES2 systems this parameter is ignored.

[JOBNO=({*low-job-no*},{*high-job- no*},{*resident-jcts*})]
defines the range of job numbers that are available for use by BDT and the number of job control tables (JCTs) that BDT is to keep resident in virtual storage.

low-job-no
specifies the lowest job number that BDT may use.

Specified As: A decimal number from 1 to 9995. To avoid wasting virtual storage, use a value of 1.

Default: 1.

high-job-no
specifies the highest job number that BDT may use.

Specified As: A decimal number from 5 to 9999. It must be at least five greater than *low-job-no*, so that at least five numbers are in the range.

Default: 1000.

resident-jcts
specifies the number of job control table (JCT) control blocks that BDT is to keep resident in the BDT address space.

Each resident JCT occupies 256 bytes of storage and resides in the JCTB cell pool. The advantage of keeping a JCT resident is that BDT need not access the BDT work queue each time it needs information that is in the JCT. On the other hand, a resident JCT occupies virtual storage for the

life of the job it represents. To help you decide whether to make JCTs resident, consider the following example.

Assume that you have requested BDT to keep up to 150 JCTs resident. Each time BDT places a job on the work queue, BDT also builds a JCT for the job and makes the JCT resident. That JCT remains resident until the job completes. If, at some time, there are 150 jobs on the BDT work queue, there will be 150 JCTs resident in virtual storage. If another job enters the system during this time, BDT will allocate a JCT to the job but that JCT will not become resident until one of the original 150 jobs complete.

If your primary concern is to conserve virtual storage, then do not request resident JCTs.

Specified As: A decimal number from 0 to 9999.

Default: 0.

Related Parameters: If you do request resident JCTs, be sure to allocate a cell pool (using the CELLPOOL statement with ID=JCTB) that is large enough. The cell pool must be large enough to contain at least five more JCTs than you request be kept resident.

[JOBRETPD=*retention-period*]
specifies the number of days that BDT may keep a job on the work queue.

Specified As: A decimal number from 0 to 365, indicating days.

Default: 0 (which means that jobs will remain on the work queue until they execute).

It is possible for a job to remain on the work queue indefinitely. This would happen, for example, if a user submitted a transaction, put the resulting job into hold status, and then forgot to release the job. To prevent jobs from remaining on the work queue indefinitely, specify a retention period other than the default of 0. (BDT considers a day to end at midnight.) BDT removes any job from the queue that has been there longer than the specified retention period.

[LOGCLASS=*print-class*]
defines the SYSOUT class to which BDT is to log its messages.

Specified As: A letter or a single-digit number. This value must also be defined as a SYSOUT class for your installation.

Default: A.

[LOGLIMIT=*line-limit*]
defines the maximum number of lines that BDT may write to the message log data set before it must print (“spin off”) that data set.

Specified As: A decimal number from 3 to 999999.

Default: 999999.

[LOGPAGE=*number-of-lines*]
defines the number of lines that are to appear on each page of the printed output of the message log.

Specified As: A decimal number from 3 to 150.

Default: 60.

[MAXTRAN=*concurrent-transfers*]
defines the maximum number of concurrent data transfers in which the home node, when it is defined as the global node, may take part. Concurrent data transfer refers to the total number of data transfers that may take place at the same time between the home node (where it is defined as the global node) and any or all of its remote nodes (where they are defined as local nodes).

Specified As: A decimal number from 1 to 999.

Default: 64.

If you are a JES3 customer doing both file-to-file and SNA NJE data transfers, *concurrent-transfers* is the total number of concurrent file-to-file and SNA NJE transfers.

Related Parameters: Enough function control tables (FCTs) to accommodate all concurrent transfers must be defined by a CELLPOOL statement (ID=FCT). One FCT is required for each active file-to-file transaction, whether the home node is global or local.

For sessions where the home node is the local node, the number of VLUs that are varied online and are active limits the number of concurrent data transfers.

[MSGPROP={YES|NO}]
determines whether BDT will route messages to nodes in the routing table if the messages come from another BDT system. For example, if system A routes messages to system B and system B routes messages to system C (or even back to system A), the MSGPROP option statement will control whether a message coming from system A is routed to system C by system B.

YES
instructs BDT to route messages coming from another BDT to nodes in the route table.

NO
instructs BDT to bypass message routing for messages coming from another BDT.

Default: YES.

[SYSLOG=({JES3|PRINT|WTO})]
identifies the output medium on which BDT is to log its messages.

Specified As: None, one, two, or all three of the following parameters. If you code only one parameter, you may omit the parentheses.

JES3
instructs BDT to log its messages on JES3 consoles that have been assigned a destination code of D22. JES2 installations should not use this parameter.

PRINT
instructs BDT to log its messages on a data set that MVS has assigned to the SYSOUT class defined by the LOGCLASS parameter. BDT prints this data when the total number of lines written to the data set matches the value specified on the LOGLIMIT parameter.

WTO
instructs BDT to log its messages on the operator's console.

Default: PRINT.

[SYSMSG={YES|NO}]
determines whether BDT logs MVS messages that have been issued by BDT dynamic application programs (DAPs). BDT logs these messages on the output medium identified by the SYSLOG parameter.

YES
instructs BDT to log DAP messages.

NO
prevents logging of DAP messages.

Default: NO.

[TQIAUTO={YES|NO}]
If TQI encounters an unrecoverable error while reading the TQI checkpoint data set, TQI disables itself. Transactions that are submitted while TQI is disabled and not required are not written onto the TQI checkpoint data set. They are, however, put onto the BDT work queue so they can be selected for execution. If one of these transactions is lost before it gets to the BDT work queue, the user must resubmit the transaction; it is not resubmitted automatically.

YES
instructs BDT to execute transactions while TQI is disabled.

NO
instructs BDT to process transactions only when TQI is working. Then if TQI encounters an unrecoverable error, it will reject transactions.

Default: YES.

[TQITIME=*read-frequency*]
specifies how often, in seconds, that BDT is to read commands and file-to-file transactions from the TQI checkpoint data set.

Specified As: A decimal number from 1 to 99999.

Default: 30 (which specifies that BDT will read from the TQI checkpoint data set every 30 seconds).

[URSCNT=*threshold*]
specifies the number of times BDT will reschedule a transaction that is waiting for an unavailable resource (URS).

threshold
defines the maximum number of times BDT will reschedule a transaction that is waiting for an unavailable resource. When BDT reaches *threshold*, it will purge the transaction. During a BDT session, the *threshold* at the global node is the *threshold* used for nodes in a global-local relationship.

Specified As: A decimal number from 0 to 255.

Default: 0 (which means that the transaction will remain on the work queue until the system can process it).

[WANTDUMP={ASK|YES|NO}]
determines whether BDT automatically dumps the BDT address space after an abnormal termination or whether the operator is given the choice of taking a dump. You may need a storage dump to determine the cause of the abnormal termination. It could be a problem within BDT itself or in one of the exit routines that you have written.

ASK
instructs BDT to ask the operator (via message BDT9990), after an abnormal termination occurs, if a dump is to be taken. The operator may select a formatted dump, select an unformatted dump, take the type of dump specified on the DUMP parameter of this OPTIONS statement, or decline a dump.

YES
instructs BDT to automatically take a dump. The DUMP parameter of this OPTIONS statement determines the type of dump taken.

NO
instructs BDT not to take a dump.

Default: YES.

Related Parameters: The DUMP parameter on this OPTIONS statement determines the type of dump taken if WANTDUMP=ASK or WANTDUMP=YES is specified.

SNABUF—Define Data Buffers

Use the SNABUF statement to allocate storage for data buffers.

Each BDT subsystem uses data buffers to send data to and receive data from other BDT subsystems. You must define the size of these buffers and the number that BDT is to allocate. You may define several different sizes. You must ensure that each buffer size you define matches the size of an VTAM request unit (RU). BDT allocates these buffers from subpools 2, 3, 4, or 5.

How Many Allowed: 1-unlimited.

How Many Required: You must code one SNABUF statement for each buffer size you define using the BUFSZ parameter of BDTNODE statements. (Thus, if you have two BDTNODE statements and each specifies the same BUFSZ value, you need only one SNABUF statement; but if each specifies a different BUFSZ value you need two SNABUF statements.) At least one SNABUF statement is required.

Placement: Between the ENDRBAM and ENDINIT statements, before or after the BDTNODE and SYSID statements.

SNABUF Statement

Defaults:

SNABUF,PRI=*num-buffs*,SIZE=*data-size*

[,ATF=*percent*]
20

[,AUTODEL={YES|NO}]
NO

[,SEC=(*num-buffs*[,*num-allocations*])]
0,1

PRI=*num-buffs* (pertains to primary buffers)
is the number of primary buffers that BDT is to allocate. BDT allocates primary data buffers during BDT initialization. The primary buffers remain allocated for the life of the BDT address space.

Specified As: A decimal number from 1 to 1000.

Default: None.

Related Parameters: The total number of buffers specified on SNABUF, that is, primary buffers (PRI parameter) plus secondary buffers (SEC parameter), must be greater than or equal to the number specified on the BDTNODE statement's BUFNO parameter.

SIZE=*data-size* (pertains to primary buffers)
is the size, in bytes, of the data portion of the primary buffers.

Specified As: A decimal number from 1 to 4096. If you specify a number that is not a multiple of 4, BDT rounds the number to the next higher multiple of 4. For example, if you code SIZE=3, BDT rounds 3 to 4; if you code SIZE=102, BDT rounds 102 to 104.

Default: None.

Related Parameters: *data-size* must be equal to or larger than the size specified (or defaulted) on the BDTNODE statement's BUFSZ parameter. If it is not, initialization will terminate.

The control portion of the primary buffers is fixed at 20 bytes. Therefore, the total size of a primary buffer, as calculated by BDT, is *data-size* (or the next higher multiple of 4) + 20 bytes. In the previous example, the allocated buffer size would be 124 bytes (104 + 20).

[ATF=*percent*] (pertains to primary and secondary buffers)
is the percentage of primary data buffers that must be free (available for use) before BDT may delete unused secondary data buffers. This parameter provides a way to reduce and possibly prevent thrashing.

Buffer thrashing is the frequently-repeated allocation and freeing of secondary data buffers. Thrashing occurs when there are frequent demands for buffers that cannot be satisfied from primary buffer storage. To satisfy the demand, BDT allocates secondary data buffers, uses them, and then frees them. Shortly thereafter, the demand for buffers, the allocation of secondary buffers, and the freeing of the secondary buffers is repeated. One solution to this problem is for you to manage buffer thrashing by specifying the ATF parameter.

Specified As: A decimal number from 5 to 100.

Default: 20.

Related Parameters: If you code the ATF parameter you must also code the AUTODEL=YES and SEC parameters on this SNABUF statement.

[AUTODEL={YES|NO}] (pertains to secondary buffers)
can be used to free any unused secondary data buffers. Freeing an unused buffer frees the storage that the buffer occupied. BDT may then use that storage for other purposes.

YES
instructs BDT to free unused secondary data buffers. If you periodically use different buffer sizes under varying loads you should code AUTODEL=YES.

NO
instructs BDT to keep unused secondary data buffers available.

Default: NO.

Related Parameters: If you specify AUTODEL=YES you must also specify the SEC parameter on this SNABUF statement.

[SEC=(*num-buffs*[,*num-allocations*])] (pertains to secondary buffers)
For each different buffer size you may optionally request that BDT allocate secondary data buffers. BDT allocates secondary data buffers only after the primary buffers of that size or previously allocated secondary buffers of that size are all in use.

num-buffs
defines the number of secondary data buffers that BDT may allocate. BDT will allocate this number of buffers each time it allocates secondary data buffers.

Specified As: A decimal number from 0 to 1000. If this is the only SEC variable you code, you may omit the parentheses.

Default: 0.

num-allocations
defines how often BDT may allocate secondary data buffers.

Specified As: A decimal number from 1 to 100.

Default: 1.

Related Parameters: The total number of buffers specified on SNABUF, that is, primary buffers (PRI parameter) plus secondary buffers (SEC parameter), must be greater than or equal to the number specified on the BDTNODE statement's BUFNO parameter.

SYSID—Name the Home Node

The SYSID statement specifies the name of the home file-to-file node or the name of the home SNA NJE node, or both. It also provides the application name and password that identify each node to VTAM.

How Many Allowed: One per initialization stream.

How Many Required: One.

Placement: Between the ENDRBAM and ENDINIT statements, before or after the BDTNODE and SNABUF statements.

SYSID Statement

Defaults:

SYSID

If this BDT system will handle file-to-file transfers:

,NAME=*node-name*,APPLID=*appl-name*

[,APPLPSWD=*vtam-password*]
8 blanks

If this BDT system will handle SNA NJE transfers:

,NJENAME=*node-name*,NJEAPPL=*lu-name*

[,NJEAPSWD=*vtam-password*]
8 blanks

NAME=*node-name* (pertains to file-to-file)
specifies the name of the home node for file-to-file transfers.

Specified As: One to eight alphanumeric characters, the first of which must be alphabetic.

Default: None.

Related Parameters: *node-name* must match the name on the N parameter of the BDTNODE statement for the home node.

APPLID=*appl-name* (pertains to file-to-file)
specifies the name by which VTAM recognizes the home file-to-file node.

Specified As: One to eight alphanumeric characters.

Default: None.

Related Parameters: *appl-name* must match the name on the ACBNAME operand of the home node's VTAM APPL statement. If ACBNAME has not been coded, *appl-name* must then match the name (label) of the APPL statement.

[APPLPSWD=*vtam-password*] (pertains to file-to-file)
specifies the VTAM password.

Specified As: One to eight alphanumeric characters.

Default: Eight blank characters.

Related Parameters: *vtam-password* must match the password specified on the PRTCT operand of the VTAM APPL definition statement.

NJENAME=*node-name* (pertains to SNA NJE)
specifies the name of the home node for SNA NJE transfers.

Specified As: One to eight alphanumeric characters.

Default: None.

Related Parameters: *node-name* must match the name on the N parameter of a remote node's BDTNODE statement (that is, a BDTNODE statement specified in a remote node's initialization stream).

NJEAPPL=*lu-name* (pertains to SNA NJE)
specifies the name by which VTAM recognizes the home SNA NJE node.

Specified As: One to eight alphanumeric characters.

Default: None.

Related Parameters: *lu-name* must match the name (label) of a VTAM APPL definition statement.

[NJEAPSWD=*vtam-password*] (pertains to SNA NJE)
specifies the VTAM password.

Specified As: One to eight alphanumeric characters.

Default: Eight blank characters.

Related Parameters: *vtam-password* must match the password specified on the PRTCT operand of the VTAM APPL definition statement.

Initialization Statement Parameters That the Operator Can Override

During BDT operation the operator can override several initialization statement parameters by issuing BDT commands. Table 6 on page 60 identifies these parameters and commands.

If the operator restarts BDT, parameter values previously set by the operator will be replaced by parameter values from the initialization stream:

- If the restart type is hot, BDT uses parameter values from the initialization stream last read.
- If the restart type is warm or cold, BDT reads the initialization stream to obtain parameter values.

Table 6. Initialization Statement Parameters That the Operator Can Override

Initialization Statement and Parameter	Overriding Command	Explanation
BDTNODE,A=	CANCEL,SNA,NODE=	The operator can cancel a session.
BDTNODE,A=	VARY,*node-name*	The operator at the global node can terminate or reestablish a session.
BDTNODE,ASR=	RESTART,SNA,NODE=	The operator can enable automatic session restart.
BDTNODE,LU=	MODIFY,NODE=,FENCE=	The operator can change the number of fenced VLUs for file-to-file transfers.
BDTNODE,LU=	VARY,*vlu-name*	The operator at the global node can change the number of available VLUs.
OPTIONS,WANTDUMP=	MODIFY,DUMP	The operator can change the dump options.
OPTIONS,SYSLOG=	MODIFY,LOG,ADEST or MODIFY,LOG,DDEST	The operator can add consoles to the list of destinations for the system log. The operator can delete previously added destinations from the list.
OPTIONS,SYSLOG=	MODIFY,LOG,FLUSH	If one of the SYSLOG parameters is PRINT, the operator can print the system log.
OPTIONS,SYSLOG=	MODIFY,LOG,SYSLOG=	The operator can change the destination of the system log.
OPTIONS,MAXTRAN=	START,SNA,LIMIT	The operator can change the limit on the total number of concurrent data transfers in which the home node can take part.
OPTIONS,TQITIME=	START,TQI,DELAY=	The operator can change the frequency with which BDT reads the TQI checkpoint data set.
OPTIONS,JES3=	VARY,JES3	The operator can enable or disable the communication path between JES3 and BDT.

Chapter 8. Writing BDT and TQI Start Procedures

This chapter describes how to:

- Write a procedure to start BDT
- Write a procedure to start the transaction queuing integrity (TQI) facility.

Step 1. Write a BDT Start Procedure

Before an operator can start BDT you must provide a start procedure. The start procedure identifies the BDT program that is to run first and the data sets that BDT is to use. Invoking the start procedure creates a BDT address space.

You must store the start procedure as a member of SYS1.PROCLIB or as a member of a data set that is concatenated to SYS1.PROCLIB. You may give the member any name you wish. There is no naming restriction like there is for the TQI start procedure.

Information about invoking the start procedure that you write is in *z/OS BDT Commands*. A discussion of cold, warm, and hot starting is included in that book. Note that starting BDT for the first time after it is installed requires a cold start.

IBM provides a sample BDT start procedure in SYS1.SBDTSAMP (member name BDT$V2SP). It is shown in Figure 23 on page 61. A description of each statement that appears in the example follows the figure.

```
//BDTA1    PROC
//BDT      EXEC PGM=BDTINTK,REGION=5000K,TIME=1440
//STEPLIB  DD DISP=SHR,DSN=SYS1.SBDTLIB
//BDSPOOL  DD DISP=OLD,DSN=BDT1.BDTSPOOL
//CRSPOOL  DD DISP=OLD,DSN=BDT1.BDTSPOOL
//DATAFILE DD DISP=SHR,DSN=BDT1.TQIDATA
//BITMAPS  DD DISP=SHR,DSN=BDT1.TQIBITS
//BDTM001  DD DISP=SHR,DSN=BDT1.MSG0001
//BDTOUT   DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//BDTABEND DD SYSOUT=A
//BDTIN    DD DISP=SHR,DSN=BDT.INISH.DECKS(BDT$FTF)
```

Figure 23. The BDT Start Procedure in SYS1.SBDTSAMP Member BDT$V2SP

//BDTA1 PROC
This statement is required. However, the procedure name can be a name other than BDTA1.

//BDT EXEC
This statement is required. You must code PGM=BDTINTK. REGION and TIME are optional.

//STEPLIB DD
specifies that BDTINTK is in SYS1.SBDTLIB.

//BDSPOOL DD
//CRSPOOL DD
These two statements define the BDT work queue data set and are required to be in the start procedure. You may not use the DYNALLOC initialization statement instead of these DD statements to allocate the data set. The ddnames must be BDSPOOL and CRSPOOL and the dispositions must be OLD. The data set name you use must be the same one you specified when you allocated the data set in “Step 2. Allocate a Data Set for the BDT Work Queue” on page 26.

//BDTM001 DD
Each message data set used by your BDT subsystem requires a DD statement in the start procedure or a DYNALLOC statement in the initialization stream. Valid ddnames are BDTM*x*, where *x* is any 1 to 4 alphanumeric characters. A suggestion is to start with the name BDTM001 and proceed sequentially: BDTM001, BDTM002, BDTM003, and so forth. The data set names you use must be the same ones you specified when you allocated the data sets in “Step 7. Allocate Message Data Sets” on page 29. The disposition of the data sets must be SHR.

//BITMAPS DD
This statement is optional. It is required if any processors in a complex will run TQI. It identifies the TQI bit-map data set. The ddname must be BITMAPS. The data set name must be the same one you specified when you allocated the data set in "Step 6. Allocate the TQI Bit-Map Data Set" on page 29. The disposition of the data set must be SHR.

//DATAFILE DD
This statement is optional. It is required if any processors in a complex will run TQI. It identifies the TQI checkpoint data set. The ddname must be DATAFILE. The data set name must be the same one you specified when you allocated the data set in "Step 5. Allocate the TQI Checkpoint Data Set" on page 28. The disposition of the data set must be SHR.

//BDTOUT DD
This statement identifies the data set to which BDT will write initialization statements and initialization messages. It is required to be in the start procedure. You may not use the DYNALLOC initialization statement instead of BDTOUT to allocate the data set.

//SYSABEND DD
This statement is optional. If you want a formatted MVS storage dump in the event BDT abnormally terminates, include this statement or a SYSUDUMP DD statement. To get an unformatted dump, include a SYSMDUMP DD statement.

To determine where the dump is to be sent, use the DUMP parameter of the OPTIONS initialization statement.

//BDTABEND DD
This statement is optional. It defines the data set to which BDT is to write the formatted dump.

//BDTIN DD
This statement identifies the data set that contains the BDT initialization stream. This statement is required to be in the start procedure. You may not use the DYNALLOC initialization statement instead of BDTIN to allocate the data set.

By means of the data set name you specify on the BDTIN DD statement, the operator can be given flexibility in choosing which initialization stream to use. When starting BDT, the operator can reply to message BDT3037 in any of the following ways:

- If the BDTIN DD statement specifies a sequential data set, the operator can reply N (for normal) to use the initialization stream in that data set.
- If the BDTIN DD statement specifies a partitioned data set without a member name, the operator can reply M=*member* to use the initialization stream in that member, or M=*xx* to use the initialization stream in member BDTIN*xx*, where *xx* is any one or two alphanumeric characters.
- If the BDTIN DD statement specifies a partitioned data set with a member name, the operator can reply N to use the initialization stream in that member, M=*member* to use the initialization stream in *member*, or M=*xx* to use the initialization stream in member BDTIN*xx*, where *xx* is any one or two alphanumeric characters.

If you have a system GMJD library (allocated in "Step 3. Allocate a System GMJD Library (File-to-File Customers Only)" on page 26) you must include a DD statement for it in the start procedure or a DYNALLOC statement for it in the initialization stream. The ddname must be GMJDLIB.

Step 2. Write a TQI Start Procedure

Before an operator can start TQI you must provide a start procedure. The start procedure identifies the TQI program, the parameters to be passed to that program, and the data sets that TQI is to use. Invoking the start procedure creates a TQI address space.

The start procedure must be available to each processor on which the TQI address space will be started. You must store the start procedure as a member of SYS1.PROCLIB or as a member of a data set that is concatenated to SYS1.PROCLIB. You must name the member TQI*x*, where *x* is the 1- to 4-character BDT subsystem name that you assigned in SYS1.PARMLIB member IEFSSN*xx* in "Step 1. Define BDT As an MVS Secondary Subsystem—SYS1.PARMLIB Member IEFSSNxx" on page 15 .

Figure 24 on page 63 shows the sample TQI start procedure that is in member BDT$V2TP in SYS1.SAMPLIB. A description of each statement that appears in the example follows the figure.

```
//TQIA1     PROC
//TQI       EXEC PGM=BDTTQIAS,TIME=1440,PARM='SYSA1,04'
//STEPLIB   DD  DISP=SHR,DSN=SYS1.SBDTLIB
//DATAFILE  DD  DISP=SHR,DSN=BDT1.TQIDATA
//BITMAPS   DD  DISP=SHR,DSN=BDT1.TQIBITS
//MESSAGE   DD  DISP=SHR,DSN=BDT1.MSG0001
//SYSUDUMP  DD  SYSOUT=A
```

Figure 24. The TQI Start Procedure in SYS1.SBDTSAMP Member BDT$V2TP

//TQIA1 PROC
This statement is required. However, the procedure name can be a name other than TQIA1.

//TQI EXEC
This statement is required. You must code PGM=BDTTQIAS and TIME=1440 exactly as shown. The PARM keyword contains one required and one optional parameter:

node-name
identifies the BDT node that will process the requests (commands and file-to-file transactions) that are on the TQI checkpoint data set. This must be the same name that you coded on one of the following SYSID initialization statement parameters (in "SYSID—Name the Home Node" on page 58.

- The NAME parameter if the BDT subsystem has a file-to-file node, without or in addition to a SNA NJE node
- The NJENAME parameter if the BDT subsystem has only a SNA NJE node.

It is also the same name that you coded on the BUILD statement used to format the TQI checkpoint and bitmap data sets (in "Step 1. Format the TQI Checkpoint, Bit-Map, and Message Data Sets" on page 31. This is a required parameter. The sample uses SYSA1.

seconds
specifies how frequently, in seconds, TQI is to read the message data set. This frequency determines how long users will have to wait to see their messages. You may select a different frequency for each TQI address space. *seconds* is optional and may be a number from 1 to 99. The default is 5. The sample uses 4.

//STEPLIB DD
specifies that BDTTQIAS is in SYS1.SBDTLIB.

//DATAFILE DD
This statement is required. It identifies the TQI checkpoint data set. The ddname must be DATAFILE. DSN must specify the name that you gave this data set when you allocated it in "Step 5. Allocate the TQI Checkpoint Data Set" on page 28. The disposition of the data set must be SHR.

//BITMAPS DD
This statement is required. It identifies the TQI bit-map data set. The ddname must be BITMAPS. DSN must specify the name that you gave this data set when you allocated it in "Step 6. Allocate the TQI Bit-Map Data Set" on page 29. The disposition of the data set must be SHR.

//MESSAGE DD
This statement is required. It identifies the message data set. The ddname must be MESSAGE. DSN must specify the name that you gave to this data set when you allocated it in "Step 7. Allocate Message Data Sets" on page 29. The disposition of the data set must be SHR.

//SYSUDUMP DD
This statement is optional. If you want a formatted storage dump in the event that TQI abnormally terminates, you must include this statement or a SYSABEND DD statement. If you want an unformatted dump, include a SYSMDUMP DD statement.

Chapter 9. Writing User Exit Routines

Exit routines are intended programming interfaces.

BDT user-written exit routines fall into two categories:

- *Authorization exit routines*, which control who may send and receive commands and file-to-file transactions. You must code these exit routines.
- *Customization exit routines*, which can alter initialization, command processing, transaction processing, and message processing. You can code these exit routines if you want; they are not required.

This chapter describes the steps in writing BDT user exit routines:

- Understanding which authorization exit routines you must write
- Deciding whether you want to write customization exit routines
- Coding the exit routines
- Assembling the exit routines
- Link-editing the exit routines
- Loading the exit routines.

Step 1. Understand Which Authorization Exit Routines You Must Write

In order to ensure that only authorized users at your node can send commands and file-to-file transactions, you must code the authorization exit routines. Even if you decide you do not need to use all of the authorization exit routines you must still code them to at least send a return code back to the calling BDT module authorizing the commands and transactions. Otherwise, BDT will fail the commands and transactions.

If you want to test BDT before coding your own authorization exit routines you can do so by using the sample exit routines in SYS1.SBDTSAMP. But you must realize that these sample routines are provided for testing purposes only; they allow BDT to run, but they do not enforce the security policies at your site.

Authorization Exit Routine in the Link Pack Area

The following authorization exit routine runs in the link pack area:

- BDTUX28—authorizes users issuing BDT commands at an MCS console. Based on that authorization level, other authorization exit routines can determine if work requests from an MCS console will be accepted.

Authorization Exit Routines in the BDT Address Space

Five authorization exit routines run in the BDT address space. Their function is to examine each command and file-to-file transaction, to determine whether the user is allowed to issue it, and to send an appropriate return code back to the calling module. These exit routine are:

- BDTUX25—authorizes users submitting commands to the BDT address space. If TQI is not enabled, it also authorizes users submitting file-to-file transactions. BDTUX25 runs when a command or file-to-file transaction enters the BDT address space. If the exit routine is not coded, BDT rejects the command or transaction. If TQI is enabled, transactions skip BDTUX25 and use BDTUX29 instead.
- BDTUX26—authorizes users submitting file-to-file transactions at the global node. BDTUX26 runs in the global BDT node before a BDT job number is assigned and the job is placed on the BDT work queue.
- BDTUX27—authorizes users submitting file-to-file transactions at local BDT nodes. If coded, it runs on both ends of a transfer just before the dynamic application programs (DAPs) for the transfer are

scheduled. The assignment of the DAPs on both ends of a transfer is the final step before the actual transfer of the data. BDTUX27 runs on both sides to make authorization checks before the transfer is actually made.

- BDTUX29—authorizes users submitting file-to-file transactions when TQI is enabled. It runs for file-to-file transactions only, not for commands. BDTUX29 must check the authorization of the user requesting the transaction.
- BDTUX31—authorizes users to submit INQUIRY and MODIFY commands concerning specific transactions (both file-to-file and SNA NJE). Information about the owner of the transaction, the origin of the command, and the type of command entered are given to BDTUX31 to determine if information about the transaction should be displayed.

Authorization Exit Routine in the JES3 Address Space

If you are a JES3 customer you must also code the following authorization exit routine to run in the JES3 address space:

- IATUX56—authorizes users submitting JES3 commands from the BDT address space.

Step 2. Decide Whether You Want to Write Customization Exit Routines

The customization exit routines that you can write are divided into five categories:

- Exit routines to alter initialization
- Exit routines for routing messages to user-defined destinations or an installation-defined log
- Exit routines to alter transaction processing
- Exit routines to alter command processing
- Exit routines to process user-defined BSIDs.

Customization exit routines are optional. You do not have to code them.

Exit Routines to Alter Initialization

Exit routines that you can use to alter the initialization process include:

- BDTUX01—allows an installation to examine or modify data areas before initialization completes and when BDT ends.
- BDTUX02—approves user-defined initialization statements read before the ENDRBAM statement during a warm or cold start.
- BDTUX03—approves user-defined initialization statements read after the ENDRBAM statement during a warm or cold start.
- BDTUX04—approves user-defined keywords entered on the BDTNODE statement when BDTNODE defines a file-to-file node.
- BDTUX05—processes user-defined information that is approved by BDTUX04.
- BDTUX06—allows an installation to examine or modify data areas before initialization is complete and control is passed to the multifunction monitor.

Exit Routines to Alter Message Processing

BDT normally routes messages to one or more of the following destinations: the originator, the BDT message log, or all applicable SYSLOG entries.

Exit routines that you can use to alter message processing are:

- BDTUX07—recognizes and processes user-defined parameters on the MSGCLASS keyword of file-to-file transactions. An alternate destination for the message can be approved in this exit routine.

- BDTUX12—allows you to alter messages that are sent to the originator
- BDTUX14—converts the transaction origin (XOID) of a user-defined location from internal to external or external to internal format. Internal to external conversion creates a readable format of the user location, one which can be placed in a message. External to internal format converts the readable format to an internal format.
- BDTUX16—gives an installation access to messages contained in a job message log (JML) if LOG was specified on the MSGCLASS parameter of a file-to-file transaction.

Exit Routines to Alter Transaction Processing

Exit routines that you can use to alter the processing of transactions include:

- BDTUX08—processes user-defined keywords on file-to-file transactions
- BDTUX15—recognizes new keywords on the PARMS transaction parameter when processing sequential file-to-file transactions
- BDTUX17—for inbound and outbound file-to-file transactions (jobs) and outbound SNA NJE jobs, reports the starting time of the job and accesses other information from the job control table (JCT)
- BDTUX18—for inbound and outbound file-to-file transactions (jobs) and outbound SNA NJE jobs, reports the ending time of a job and accesses other information from the job control table (JCT)
- BDTUX19—makes final changes to file-to-file transaction text before the transaction definition is accepted
- BDTUX24—accesses the SMF type 59 record before it is written
- BDTUX30—associates an owner (other than BDT) with a tape or DASD data set.

Note that authorization exit routines BDTUX25, BDTUX26, BDTUX27, BDTUX28, and BDTUX29, which were discussed in “Step 1. Understand Which Authorization Exit Routines You Must Write” on page 65, are also associated with transaction processing.

Exit Routines to Alter Command Processing

An exit routine that you can use to alter command processing is:

- BDTUX10—processes passwords specified on commands. Passwords can authorize users to issue commands they are not otherwise authorized to issue.

Note that authorization exit routines BDTUX25, BDTUX28, and BDTUX31, which were discussed in “Step 1. Understand Which Authorization Exit Routines You Must Write” on page 65, are also associated with command processing.

Exit routines to recognize user-defined BSIDMOD fields

Exit routines that can recognize user-defined BSIDMOD fields include:

- IATUX50—recognizes user-defined BSIDMOD fields in BSIDs within the JES3 address space
- BDTUX11—recognizes user-defined BSIDMOD fields in BSIDs within the BDT address space.

Step 3. Code Your Exit Routines

In order to code your exit routines you will have to refer to:

- Chapter 10, “User Exit Routine Reference,” on page 81 to find out exactly when each routine receives control, what the register conventions are, and so forth
- Chapter 11, “Mapping Macro Reference,” on page 139 to find out the formats of mapping macros that you will use in your routines
- Chapter 12, “Executable Macro Reference,” on page 143 to find out the formats of executable macros that you will use in your routines.

Following are some coding considerations.

General Considerations When Writing BDT Exit Routines

The sample exit routines contained in SYS1.SBDTSAMP contain general header information that identifies each exit routine. The offset of this information as presented in each sample exit routine is critical and specific to that exit routine. You must add the header information to each exit routine you code. When you code your own routines, be certain to use the same offsets for whatever information you choose to include; the exit name must be included to provide a label to be added to a dump should one be taken. Also, this general header information is required in order that BDTGRSV can create a BDT trace entry. If this header information is not present and the module size is less than the standard exit header length, an 0C4 abend can occur. See Figure 25 on page 68 for an example of the exit routine header information.

```
************************************************************
*                                                          *
*                EXIT HEADER INFORMATION                   *
*                                                          *
************************************************************
BDTUX31   CSECT

          B   ENDCATCH-*(,R15)   BR AROUND ENTRY INFORMATION
          DC  AL1(ENDCATCH-*-1)  LENGTH OF ENTRY INFORMATION
MDNUX31   DC  CL8'BDTUX31'       MODULE NAME
MDLUV31   DC  CL8'       '       LABEL NAME
MDRUX31   DC  CL9' HBD1102 '     BDT RELEASE OR PTF NUMBER
MDDUX31   DC  CL8'06/20/84'      ASSEMBLY DATE
MDTUX31   DC  CL6'-22.35'        ASSEMBLY TIME
ENDCATCH  DS    OH
```

Figure 25. Example of Standard Exit Routine Header Information

How exit routines are invoked

BDT uses a macro, BDTXUEX, to transfer control to user exit routines. The BDTXUEX macro either calls or branches to each routine, depending on whether the routine is in the BDT address space or the link pack area:

- BDTXUEX uses the BDTXCALL linkage to reach user exit routines that are in the BDT address space. The calling module's registers are saved in BDTGRSV prior to invoking the exit routine, the exit routine is linked, and exit processing is performed. Return to the calling routine is through register 14, and includes a return to BDTGRSV to reload the calling routine's registers before returning control to the calling module. (Refer to Figure 26 on page 69, part A, for a representation of this processing.)
- BDTXUEX uses a BALR instruction to reach user exit routines that are in the link pack area (always BDTUX28 and usually BDTUX08, BDTUX10, and BDTUX19). As a result, the caller's registers are not saved, nor are the registers automatically restored upon return; this is your responsibility when coding these exit routines. The BDTXCALL linkage cannot be used in this case: the registers are not saved in BDTGRSV because the BDTXCALL linkage has no access to the BDT TVT. Return to the calling module is direct with use of a BR14 instruction. (Refer to Figure 26 on page 69, part B, for a representation of this processing.)

Note: The BDTXUEX macro considers all return codes other than a multiple of 4 to be invalid.

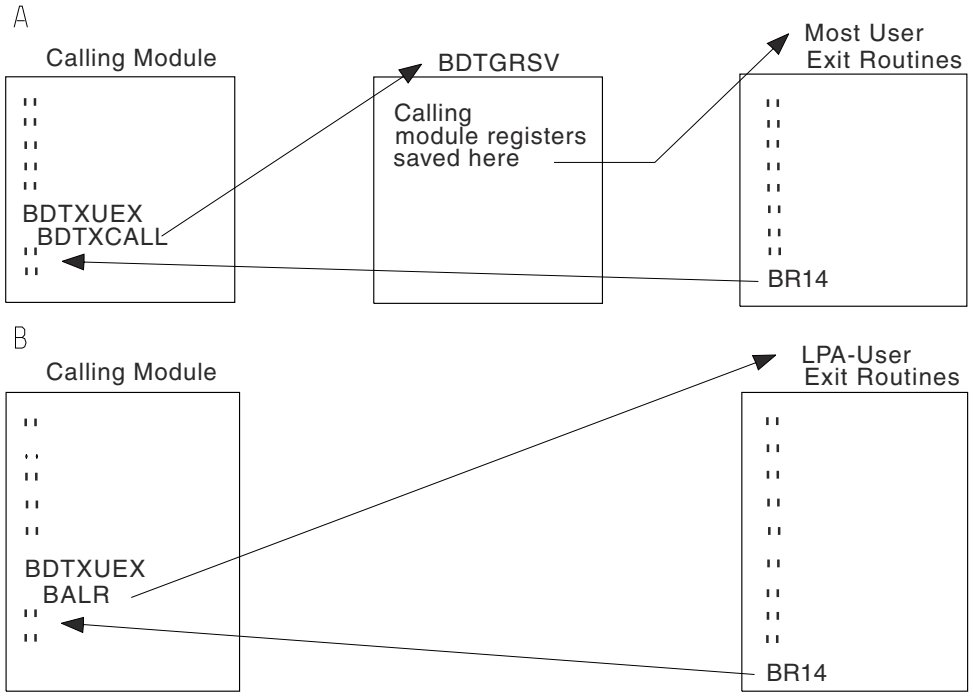


Figure 26. BDTXUEX exit routine linkage conventions

Names of Modules That Invoke the Exit Routines

The description of each user exit routine in Chapter 10, "User Exit Routine Reference," on page 81 states the name of the BDT (or JES3) module that invokes the exit routine. The following table summarizes this information.

User Exit Routine
Module That Invokes It

BDTUX01
BDTINTK

BDTUX02
BDTINIC

BDTUX03
BDTINCD

BDTUX04
BDTINR1

BDTUX05
BDTINR2

BDTUX06
BDTINIT

BDTUX07
BDTGRXD

BDTUX08
BDTLP

BDTUX10
BDTLP

BDTUX11
BDTCMDV

BDTUX12
BDTCMDV

BDTUX14
BDTGRXD

BDTUX15
BDTSEQ

BDTUX16
BDTGRLG

BDTUX17
BDTGRJS

BDTUX18
BDTGRJS

BDTUX19
BDTLP

BDTUX24
BDTACMN

BDTUX25
BDTCMDV

BDTUX26
BDTGRXD

BDTUX27
BDTGRJR

BDTUX28
BDTSS34

BDTUX29
BDTTQI

BDTUX30
BDTGRDA

BDTUX31
BDTIQDV

IATUX50 (JES3)
IATBDCI (JES3)

IATUX56 (JES3)
IATBDCI (JES3)

Using Text Units to Customize BDT Transaction Processing

For each parameter specified on a transaction, BDT builds a control block called a *text unit*. These text units describe the processing to be done for the transaction as a whole or the processing to be done specifically for the TO or the FROM data set.

BDT puts the text units in the master job definition (MJD) for the transaction. Exit routine BDTUX08 can be used to define user transaction parameters that can be used to create text units. Before a transaction is placed on the BDT work queue, exit routine BDTUX19 can be used to add text units to the MJD. BDTUX19 can also be used to inspect and modify text units already in the MJD.

BDT processes two kinds of text units:

- Dynamic allocation text units

  These text units are created from parameters specified either in the TO section or the FROM section of a transaction. (For a description of the sections of a BDT transaction, refer to *z/OS BDT File-to-File Transaction Guide*. For example, dynamic allocation text units are created for the VOLUME and the DSORG transaction parameters. Dynamic allocation text units are always considered to be *nongeneric*: that is, each pertains either to the TO data set or to the FROM data set, not to the transaction as a whole. Dynamic allocation text units are defined by MVS allocation services. BDT passes them to allocation services via SVC 99 to allocate the source and the destination data sets. See *z/OS MVS*

Programming: Authorized Assembler Services Guide for information about dynamic allocation text units in MVS.

- BDT text units

  These text units are created from parameters specified in the job definition, the TO, or the FROM section of a transaction. A BDT text unit can describe either processing for the whole transaction (a *generic* text unit) or processing specifically related to the TO or the FROM data set (a nongeneric text unit). For example, generic BDT text units are created for the JOBNAME and MSGCLASS transaction parameters. Nongeneric BDT text units are created for the LOCATION and BDTENQ transaction parameters. BDT text units are not passed to SVC 99 to allocate a data set. They are used by BDT in its processing of the transaction.

Each text unit created for a transaction is identified by a key. Macro IEFZB4D2 maps the key values for dynamic allocation text units. Macro BDTDMJD maps the key values for BDT text units. User exit routines that have access to the MJD can use the BDTXTUAM macro to retrieve a particular text unit by specifying its key. Refer to "BDTXTUAM" on page 157 for a description of how to use the BDTXTUAM macro.

You can create your own transaction parameters and associated text units that specify processing unique to your installation. This can be done by using the BDTDKYWD macro in BDTUX08, the language processor keyword extension table. This table extends BDT's keyword table that defines valid transaction parameters and the text unit table that describes the corresponding text unit for each keyword. Note that BDTUX08 does not contain executable code; its purpose is to add entries to the keyword and text unit tables. Refer to the description of BDTUX08 in "BDTUX08—User-Defined File-to-File Transaction Keywords" on page 97 and the description of the BDTDKYWD macro in "BDTDKYWD" on page 143.

You can also add text units to the MJD during transaction processing with BDTUX19, the transaction modification exit routine. BDTUX19 receives control from the language processor after it has processed all available transaction parameters. (Note that BDTUX19 will receive control twice: once in the user's address space (or in BDT's address space for some transactions entered at MCS consoles), and again in BDT's address space after the language processor has processed the parameters specified in the GMJD library.) In BDTUX19 you may process the text units created in BDTUX08 for your installation-unique parameters. You may also inspect, modify, or add any other text units. See the description of BDTUX19 in "BDTUX19—File-to-File Transaction Modification" on page 116 for a discussion of the functions this exit routine may perform.

After BDTUX19 is invoked in the BDT address space to make any final changes to a transaction, the transaction is written to the BDT work queue. Subsequent user exit routines may retrieve text units via BDTXTUAM and inspect them, but they may not change existing text units or add new ones.

A Short Cut for Testing BDT

The BDT authorization exit routines must be coded to authorize a user to issue BDT commands and transactions. However, if you want to test BDT, there are two quick methods of doing so. First, you can use the sample authorization user exit routines provided in SYS1.SBDTSAMP. The sample exit routines in SYS1.SBDTSAMP allow transactions to be submitted from **all** sources (that is, batch, TSO, and MCS and JES consoles); however, commands can only be submitted through MCS and JES consoles (batch and TSO are not supported). The second method is to code your own user exit routines to set a return code and return to the invoking module.

You can code each authorization exit routine to set a return code which, when passed back to the invoking module, causes the authorization of the transaction or command. BDT authorizes work requests when a return code of 0 is received from a user exit routine. A return code of 4 in the JES3 address space from IATUX56 authorizes the JES3 command entered by BDT.

These shortcuts should *never* be taken when BDT is installed on a production system. Authorizing all commands and transactions can seriously jeopardize system security.

How Authorization Exit Routines Fit into the Flow in a BDT File-to-File Subsystem

BDT Request Routing

BDT processes two types of requests from file-to-file users:

- A request for a file transfer. The data is moved from one data set to another. This is called a transaction.
- A request for BDT to perform some function. This function is to either provide information, as with an inquiry command, or to change the status of its environment in some way. For example, this may be a request to schedule work requests or the use of devices. This is called a command.

In a JES3 environment BDT also transmits JES3 commands. In this case BDT merely acts as a route for the command from you to the JES3 address space. IATUX56 serves this function.

How Requests Enter a BDT File-to-File Subsystem

Commands and file-to-file transactions can enter the BDT subsystem from four sources: JES3 consoles (only in a JES3 environment), MCS consoles, TSO terminals, or batch jobs. Figure 27 on page 73 shows these four sources of input. Note that the routing of commands and file-to-file transactions is affected by an installation's use of transaction queuing integrity (TQI) to checkpoint the work request at the processor on which it is entered.

BDT TQI is a facility that ensures that commands and file-to-file transactions are not lost in the event of hardware or software failure. Once started, TQI must be enabled to write those commands and transactions to the TQI files. BDT TQI ENABLE commands must be entered at an MCS console.

Each of the two types of work requests, commands and transactions, takes a different path to execution. Each needs authorization checks along the way to confirm that the user of the command or transaction has the authority to be using it. Authorization user exit routines must check the input before the command or transaction has left the node on which it is entered. The node receiving the work request must also verify the authority of the user before executing the work. For transaction-specific inquiry (I) and modify (F) commands, BDTUX31 is provided immediately before processing a specific transaction, to authorize or reject that command based on a comparison of the transaction origin and the command origin.

BDT Requests from a TSO Terminal

As you will note in Figure 27 on page 73, after a TSO user enters a BDT request, it is processed by the BDT command processor, BDTTSO. BDTTSO issues an SVC to route it to IGX00034. This module invokes the BDT language processor (BDTLP), building the first control block, called the BDT subsystem interface data area (BSID), which contains the user's request. The BSID is then passed to BDTSSBDT by the subsystem interface (SSI).

BDT Requests from a Batch Job

When a BDT request is issued by a batch job, the request is processed by a BDT batch processor (BDTBATCH) which issues an SVC to route it to IGX00034. This module invokes the BDT language processor (BDTLP). BDTLP builds the first BDT control block, called the BSID, to contain the user's request. The new BSID is passed to BDTSSBDT by the subsystem interface (SSI).

BDT Requests from an MCS Console

Note in Figure 27 on page 73 that when an MCS console operator enters a BDT request, the request is first processed by BDTSS34. User exit routine BDTUX28 in BDTSS34 can examine the MCS console to determine its attributes and, based on those attributes, assign it an authorization level. The authority level is passed back to BDTSS34 which then places the authorization level in the BSID. Later, when other authorization user exit routines examine the BSID and its authorization level, authorization user exit routines can determine if those requests should be processed.

A skeletal (or native mode) BSID is built and passed to BDTSSBDT. It is passed by the subsystem interface (SSI).

BDT Requests from a JES3 Console

A BDT work request entered at a JES3 console is processed by IATBDCI, a JES3 module. IATBDCI calls the language processor (BDTLP) to build the first BDT control block. This control block, called the BSID, contains the user's request. IATBDCI passes the BSID to BDTSSBDT by the subsystem interface (SSI).

Checkpointing requests in BDT TQI

Note in Figure 27 on page 73 that all four sources of work for BDT (TSO terminal, batch job, an MCS console, or a JES3 console) use TQI to save their commands or file-to-file transactions.

TQI ensures that work requests are not lost through hardware or software failures. All file-to-file transactions are always checkpointed, but commands are checkpointed only when BDT is active. A BDT SEND command is checkpointed only if a session with the node specified on the SEND command is active. TQI is a separate address space working in each processor where BDT requests are issued.

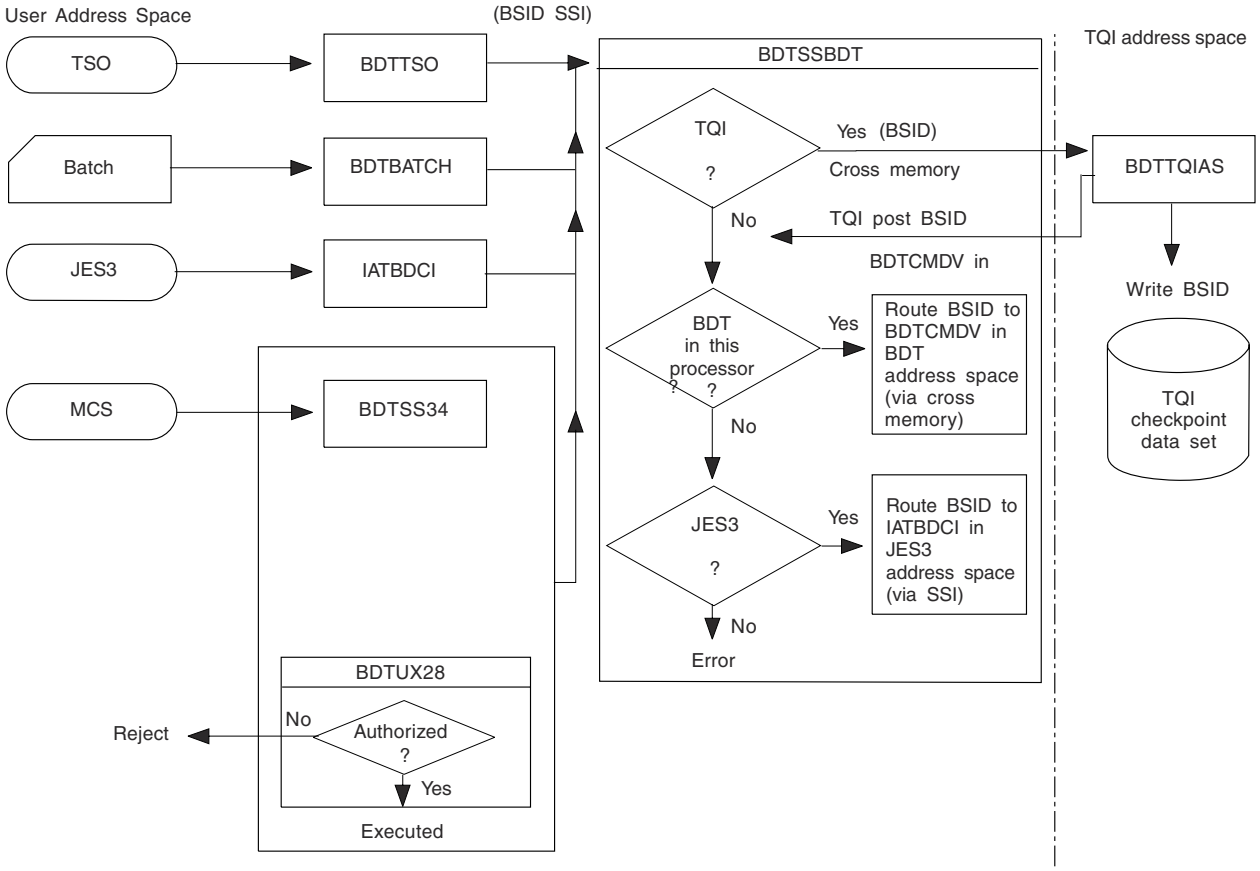


Figure 27. File-to-file request routing

Routing from BDTSSBDT to the BDT address space

Figure 28 on page 75 shows the continued processing of a BDT transaction or command from BDTSSBDT. Note that in traveling from BDTSSBDT to the TQI address space, then to BDT, the request goes from the user's address space to the BDT address space. Details on the use of each of the authorization exit routines can be found in Chapter 10, "User Exit Routine Reference," on page 81.

At this point in the logic flow, routing is dependent upon the type of request the user has made. Note in Figure 28 on page 75 that:

- If BDT is active in the processor, and TQI is not enabled and not required, the transaction or command is routed directly to the BDT address space. Otherwise, the transaction or command is rejected.
- If BDT is not active in the processor, and TQI is not enabled and not required, and the request was made on a JES3 system, the transaction or command is routed through the JES3 address space (IATBDCI).

BDTUX25 screens all commands entering the BDT address space. BDTUX25 also screens file-to-file transactions entering the BDT address space when TQI is not active and not required. If the request is checkpointed by TQI, BDTUX29 in BDTTQI processes will process transactions and BDTUX25 will process commands. Once the initial authorization check is made, routing continues in the following way:

- BDTGRXD picks up file-to-file transactions that are to be executed in this node and sends them through user exit routine BDTUX26 for another authorization check. This flow is illustrated in Figure 29 on page 77.
- BDT commands to be executed in this node complete execution. Authorization is complete.
- BDT commands to be executed at another node are examined in authorization exit routines at the other node. In the case of a BDT command sent to another node, the command is authorized in BDTUX25 on the sending node, travels across the link to the other node, and is authorized once again in BDTUX25 in the receiving node. All nodes must code the authorization exit routines to prevent unauthorized users from submitting BDT commands and transactions, not only within a node but from other nodes as well.

Figure 28 on page 75 illustrates that BDTTQI is notified of the arrival of a checkpointed request. BDTTQI then reads the request off the BDT TQI checkpoint file. If the request is a BDT command, BDTTQI sends the command to BDTCMDV for authorization checking in BDTUX25. If the request is a file-to-file transaction, BDTTQI passes it through its own authorization exit routine, BDTUX29.

If the request is to be scheduled on this node, BDTTQI passes it directly to BDTGRXD for processing. Otherwise, it passes by BDTIFCM communications to the appropriate node.

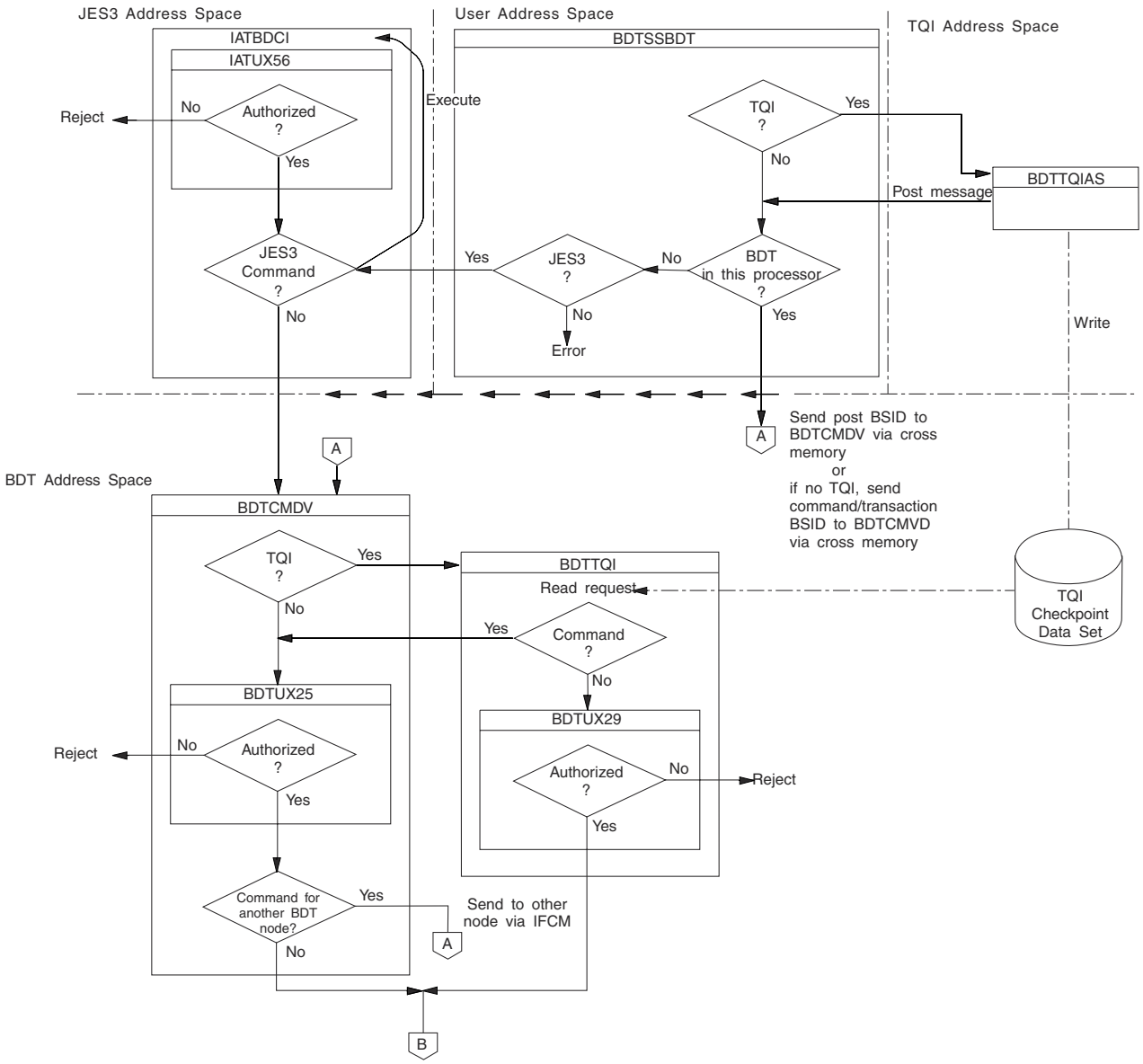


Figure 28. Routing from BDTSSBDT to BDT

Routing of a file-to-file transaction

Let's take a look at the authorization checks made on a transaction processed at this node. In Figure 28 on page 75, we left off with a transaction that completed an authorization check in BDTUX25 in BDTCMDV if BDT TQI is disabled. In the case in which BDT TQI is enabled, transactions skip BDTUX25 and are examined by user exit routine BDTUX29.

Regardless of whether a file-to-file transaction is authorized in BDTUX25 or BDTUX29, the path after that point is identical for transactions:

- If the file-to-file transaction is bound for another node, the exit routines run in the following order after passing through either BDTUX25 or BDTUX29:
  - BDTUX26 in the global node performs an authorization check. Remember that BDTUX26 runs on the global node's copy of BDTGRXD. Any limitation on data sets or prohibition on certain users in the BDT network would ideally be made at this point in processing. Otherwise, the transaction is put on the BDT job queue and subsequently scheduled for execution. After the job has been scheduled, BDTUX27 can be used to make further checks on either side of the transaction, before the DAPs receive control to make the data transfer. BDTUX26 should be used for as much authorization checking as possible; this prevents file-to-file transactions waiting on the job queue to only be

unauthorized later (that is, not being authorized by BDTUX27). Some checking must be provided in BDTUX26, however. For example, if a file-to-file transaction is submitted at the global node, BDTUX27 will be the first check to determine if the user is permitted use of that resource (for example, a unit, a volume, or a data set).

– The local node check in BDTGRJR is made on both ends of the data flow. The local authorization check on the node receiving the information is made first. Exit routine BDTUX27 in the sending node examines the file-to-file transaction after the receiving node completes its authorization.

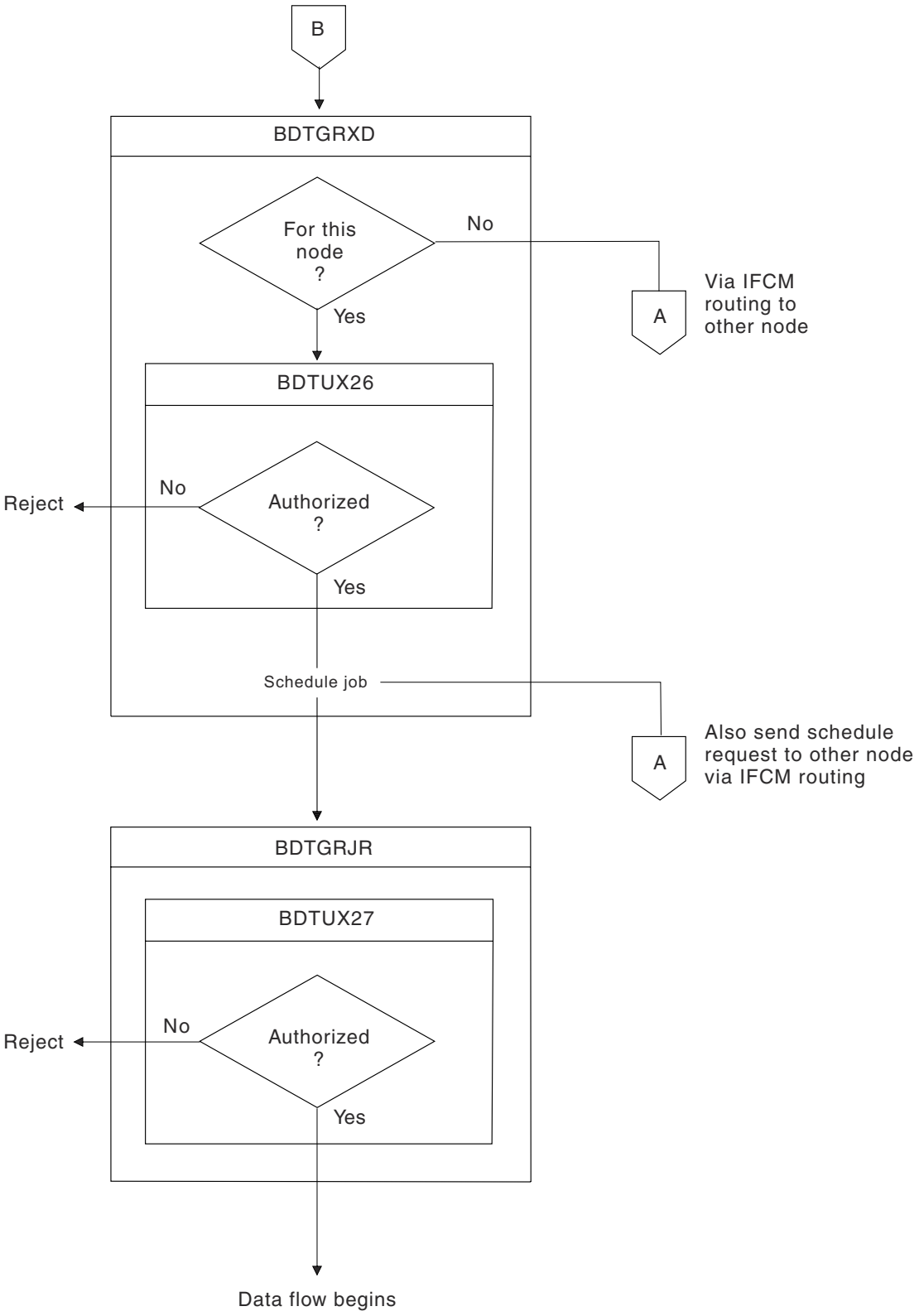


Figure 29. Routing of a file-to-file transaction

Routing of a Command

Let's take a look at the authorization checks made on a command processed at this node. In Figure 28 on page 75, a command completes the authorization check done in BDTUX25. The command is then passed to the appropriate handler routine. For commands that do not request information about specific transactions, such as an I B command, the command is processed and a response is routed back to

the requestor. Transaction-specific commands, such as I Q, are processed by another authorization exit routine, BDTUX31.

BDTUX31 is entered every time a response is generated during the processing of a command. For an I Q command attempting to display information for ten jobs on the BDT job queue, BDTUX31 is entered for each job. BDTUX31 is passed information about the origin of the command and the origin of the transaction. The exit routine can then determine if the response should be displayed or suppressed for the transaction. This prohibits users from inquiring after, or possibly modifying, jobs that they do not own.

Step 4. Assemble Your Exit Routines

You must use the High Level Assembler (HLASM).

Step 5. Link-Edit Your Exit Routines

Exit Routines That Will Run in the Link Pack Area

Exit routine BDTUX28 is invoked by CSECT BDTSS34. Both must run in the link pack area.

Exit routines BDTUX08, BDTUX10, and BDTUX19 are invoked by CSECT BDTLP. All four may run in the BDT address space or in the link pack area.

Exit Routines That Will Run in the JES3 Address Space

Exit routines IATUX50 and IATUX56 are for JES3 customers only and run in the JES3 address space. See *z/OS JES3 Customization* for information about these exit routines.

Exit Routines That Will Run in the BDT Address Space

All BDT exit routines except BDTUX28 can run in the BDT address space. You can link-edit them as members of SYS1.SBDTLIB (the BDT module library) or as members of a library that is concatenated with SYS1.SBDTLIB.

Step 6. Load Your Exit Routines

Loading Exit Routines into the Link Pack Area

BDTUX28 must be loaded into the link pack area (LPA) or Modified Link Pack Area (MLPA). The BDTLP module, containing exits BDTUX08, BDTUX10, and BDTUX19, may be placed in a link list library, a STEPLIB, or loaded into LPA or MLPA. All LPA and MLPA resident modules must be reentrant, and loading or changing them requires an IPL.

To load modules into LPA, place them in a library in the LPA list. To load modules into MLPA, list them in an IEALPAxx member of PARMLIB used for IPL. Loading new or updated modules into LPA requires an IPL with the CLPA option. Loading new or updated modules into MLPA requires an IPL. For more information about loading modules into LPA and MLPA, see *z/OS MVS Initialization and Tuning Reference*.

BDTUX08, BDTUX10, and BDTUX19 can be tested without an IPL even if the BDTLP module is loaded into LPA or MLPA. You can link a test copy of BDTLP into a different library and use STEPLIB to test the copy. You can also test these exits this way if BDTLP is in a link list library.

If BDTLP is in a link list library, BDTUX08, BDTUX10, and BDTUX19 can change without an IPL by relinking BDTLP and issuing a MODIFY LLA,REFRESH operator command.

Loading Exit Routines into the BDT Address Space

You must hot start BDT to load exit routines into the BDT address space. All user exit routines except BDTUX28 may be loaded into the BDT address space. (BDTUX28 must be in the link pack area.)

For user exit routines residing in the BDT address space, BDTINIX loads the routines using the MVS LOAD macro and stores the address of each in the user exit list in BDTGRPT. Unlike JES3, BDT does not require dummy exit routines with a branch back to the main code.

Loading Exit Routines into the JES3 Address Space

You must hot start JES3 to load exit routines into the JES3 address space. IATUX50 and IATUX56 must be loaded into the JES3 address space.

Chapter 10. User Exit Routine Reference

This chapter provides interface information about each BDT user exit routine that you can write. It provides:

- The type of routine: authorization or customization
- A general description of the routine
- Register conventions at entry to and exit from the routine
- Circumstances under which the exit routine is entered and exited
- The execution environment: the point when the routine receives control, the address space in which it runs, the task under which it runs, the PSW state, and the storage protection key
- The mapping and executable macros used by the routine
- The consequences if the routine is not used.

The required authorization exit routines (BDTUX25, BDTUX26, BDTUX27, BDTUX29 and BDTUX31) are SMP/E installable. For more information, see the prolog of the exit routines in SYS1.SBDTSAMP.

The exit routines are listed in alphanumeric order.

BDTUX01—BDT Initialization and Termination Processing

Type

Customization (optional).

General Description

This exit routine is called when BDT is started, when BDT initialization completes, and when BDT ends, to allow an installation to perform any specialized processing necessary only for the life of the BDT address space.

It should be noted that it is possible during an abnormal termination of the BDT address space, for an invocation of BDTUX01 with reason code = 16 or 20, that the TVT is inaccessible when BDTUX01 is called. Therefore, BDTUX01 uses the BDTXCALL macro to remove a dependency that BDTXUEX has on the TVT. For reason code 16 or 20, BDTUX01 will not be called via ASAVE, and no save area will be passed to the exit routine.

Also consider TVT inaccessibility if anchoring BDTUX01 data areas in the TVT.

This exit routine receives control from BDTINTK with the following information:

1. A reason code in register 0 indicates that this exit routine was invoked during one of the following conditions:
   - 0 - Pre-initialization processing
   - 4 - Initialization successfully completed
   - 8 - Termination processing when initialization is successful
   - 12 - Termination processing when initialization is not successful
   - 16 - Termination processing when initialization is successful from BDTINTK's ESTAE recovery routine
   - 20 - Termination processing when initialization is not successful from BDTINTK's ESTAE recovery routine.
2. The address of the TVT that is contained in register 12.

Note: BDTUX01 will not receive control out of BDTINTK's ESTAE routine if the ESTAE routine is entered after BDTUX01 has been invoked with a reason code of 8 or 12.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX01 routine. Registers 2-14 are saved by BDT ASAVE processing. For reason code 16 or 20, registers 2-14 are saved by BDTINTK in its own internal data area.

Register 0
Contains a reason code:

Reason 0
Pre-initialization processing

Reason 4
Initialization successfully completed

Reason 8
Termination processing when initialization is successful

Reason 12
Termination processing when initialization is not successful

Reason 16
Termination processing when initialization is successful from BDTINTK's ESTAE recovery routine

Reason 20
Termination processing when initialization is not successful from BDTINTK's ESTAE recovery routine

Register 12
Address of the BDT TVT. As previously noted, it is possible that when BDTUX01 is invoked with a reason code 16 or 20 that the TVT is inaccessible. If so, then register 12 is zero.

Register 13
Points to the register save area set up by ASAVE. If reason code = 16 or 20, register 13 is zero. BDTINTK will save its own registers prior to calling BDTUX01 and will not link up a new save area for BDTUX01.

Register 14
Contains the address of the return point in BDTGRSV. If reason code = 16 or 20, register 14 contains the return point within BDTINTK.

Register 15
Contains the entry point address into the BDTUX01 user exit routine.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX01 user exit routine, registers 2-14 of BDTINTK are saved by BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point in BDTGRSV. If reason code = 16 or 20, register 14 contains the return point within BDTINTK.

Register 15
No return codes - register 15 is ignored.

Operation

BDTINTK invokes this exit routine before initialization begins, after initialization ends, and when BDT terminates. Parameters are not passed to this exit routine and any user data areas that need processing can be anchored in the BDT TVT.

Note: When you select user-reserved fields in the TVT for storage of user data, do not use fields in the TVT initialization save area (TVTINSAV to TVTENDSV) before BDT is initialized. Data that is saved in the TVT after initialization will be saved and available during a hot or warm start.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your user exit routine. If an ESTAE exit routine is not included in the exit routine, an ESTAE in BDTABMN provides clean-up in the event of exit routine failure.

Environment

Point Where Exit Routine Receives Control: This exit routine receives control at:

- BDT pre-initialization time
- BDT post-initialization time
- BDT termination time.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under the TCB for BDTINTK.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDTVT to map the TVT
- BDTDREG to map the registers
- BDTDGSD to map the GSD (necessary for coding an ESTAE).

Executable Macros

- BDTXASRV to invoke abend services during abend recovery

What If BDTUX01 Is Not Used?

If the exit routine is not provided, processing is not affected.

BDTUX02—Unrecognized Spool Data Management (RBAM) Initialization Statements

Type

Customization (optional).

General Description

This exit routine allows an installation to identify and process spool data management (RBAM) initialization statements not recognized by BDTINIC. RBAM statements precede network initialization statements processed by BDTUX03. BDTINIC processes the OPTIONS statement and the ENDRBAM statement. The exit routine runs only during cold or warm starts. BDTINIC is the initialization module that processes initialization statements that structure the BDT work queue.

You can also use the exit routine to identify and process unrecognized keywords specified on the OPTIONS statement.

For example, you might write:

```
USERSTMT,PASS=YES
   .
   .
   .
OPTIONS,PRIORITY=6
```

where:

USERSTMT is the unrecognized user statement
PASS is the keyword on the unrecognized user statement
YES is the value associated with the keyword
OPTIONS is the recognized BDT initialization statement
PRIORITY is the unrecognized keyword on the BDT initialization statement. The value associated with the keyword is 6.

See Appendix E, “Initialization Flow and User Exit Routines,” on page 169 for an understanding of how this user exit routine fits into the initialization process.

BDTINIC invokes BDTUX02 for four reasons. A reason code in register 0 informs the user exit routine of the purpose for which it is run. The user exit routine does not parse the statement; parsing is performed by BDTINIC. BDTINIC passes the information to the user exit routine.

Reason=0 Invalid initialization statement

This invocation allows the user to recognize and process initialization statements not recognized by BDT. BDTINIC calls the exit routine to examine the statement. If the exit routine does not recognize it, BDTINIC processes the statement as an error.

Reason=4 Process a keyword on the user-defined statement

This invocation allows the user to recognize and process a keyword and its parameters on a user-defined initialization statement. BDTINIC calls BDTUX02 with reason=4 as long as there are keywords associated with a user-defined initialization statement.

Reason=8 EOD condition on a user-defined statement

This invocation communicates the end-of-data condition to the user exit routine. BDTINIC has passed all the keywords on an unrecognized statement.

Reason=12 Unrecognized keyword on an OPTIONS statement

This invocation allows the user to recognize and process user-defined keywords on the OPTIONS initialization statement.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX02 user exit routine. Registers 2-14 are saved by BDT ASAVE processing in BDTGRSV.

Register 0
Contains a reason code:

Reason 0
Unrecognized initialization statement

Reason 4
Keyword on the user-defined initialization statement

Reason 8
End-of-data condition

Reason 12
Invalid OPTIONS statement keyword

Register 1
Contains the address of the parameter list. The parameter list contents depend upon the reason for which the user exit routine runs.

Reason 0
Register 1 contains the address of a one-word parameter list:

Word 1
Address of the unrecognized initialization statement

Reason 4
Register 1 contains the address of a three-word parameter list:

Word 1
Address of the keyword on a user-defined initialization statement

Word 2
Address of the keyword parameter value

Word 3
Address of a halfword field containing the length of the keyword parameter value

Reason 8
No parameters are passed when the exit routine runs for an EOD condition.

Reason 12
Register 1 contains the address of a three-word parameter list:

Word 1
Address of the OPTIONS initialization statement unrecognized keyword

Word 2
Address of the keyword's parameter value

Word 3
Address of a halfword field containing the length of the keyword parameter value

Register 11
Contains the address of the initialization data CSECT, BDTINDT.

Register 12
Contains the address of the BDT TVT.

Register 13
Points to the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point in BDTGRSV.

Register 15
Contains the entry point address into the BDTUX02 user exit routine.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX02 user exit routine, registers 2-14 of BDTINIC are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either a GETMAINed area or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point in BDTGRSV.

Register 15
Used for returning a return code value set by the BDTUX02 user exit routine.

For reason codes 0, 4, and 8:

RC 0
Indicates that the BDTUX02 user exit routine recognizes the statement and BDTINIC continues initialization processing.

RC 4
Indicates that BDTUX02 does not recognize the statement or keyword. BDTINIC prints the statement and initialization continues until complete. At this time, BDT terminates due to the error.

RC 8
Indicates that the BDTUX02 user exit routine failed to recognize the statement or its keywords. BDTINIC prints the statement and causes initialization to terminate as soon as possible.

For reason code 12:

RC 0
Indicates that the BDTUX02 user exit routine recognizes the user-defined keyword on the OPTIONS statement and processing continues.

RC 4
Indicates that BDTUX02 does not recognize the user-defined keyword on the OPTIONS statement. BDTINIC prints an error message and processing continues until initialization is complete. At this time, BDT terminates due to the error.

RC 8
Indicates that the BDTUX02 user exit routine failed to recognize the user-defined keyword on the OPTIONS statement. BDTINIC prints an error message and terminates initialization as soon as possible.

Operation

When BDTINIC encounters an unknown initialization statement, it passes control to the user exit routine with reason code=0. If the exit routine recognizes the statement, it sets a return code of 0 in register 15 and returns to BDTINIC.

BDTINIC invokes the exit routine again for each keyword it encounters on the user-defined initialization statement. The exit routine runs in this case with reason code=4.

When all keywords have been processed, the exit routine is called with reason code=8, which informs the exit routine of the end-of-data condition on the user-defined keyword.

For each exit call with reason codes 0, 4, and 8, a return code of 0 indicates the exit routine recognizes the statement as user-defined; a return code of 4 indicates the exit routine failed to recognize the parameter as user-defined, but the error is not serious enough to fail BDT initialization immediately. A return code of 8 indicates an error serious enough to prevent BDT from completing initialization as soon as possible. Error messages are issued for return codes 4 and 8.

For reason code 12, a return code of 0 indicates that the user-defined keyword on the OPTIONS statement is recognized. Processing continues. A return code of 4 causes BDTINIC to issue an error message; processing continues until the end of initialization. Then BDT terminates. A return code of 8 causes BDTINIC to issue an error message, then terminate initialization processing as soon as possible.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your exit routine in the event of an abend. If there is no recovery in the routine, an ESTAE recovery routine in BDTABMN provides clean-up in the case of system failure.

Environment

Point Where Exit Routine Receives Control: This exit routine receives control during BDT cold or warm starts when the initialization file must be opened, read, and closed.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under the TCB of BDTINIT.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDINT to map the initialization data CSECT
- BDTDREG to map the registers
- BDTDTVT to map the TVT
- BDTDGSD to update the generalized subtask directory (GSD) (if you code an ESTAE)

Executable Macros

- BDTXASRV to invoke abend services (if you code an ESTAE)

What If BDTUX02 Is Not Used?

If the exit routine is not provided, the error handling for the unrecognized initialization statement issues error message BDT3243. It displays the unrecognized statement. The message processing sets the severe error indicator. This causes the eventual termination of BDT initialization. BDTINIC processing continues by reading the next initialization statement.

BDTUX03—Unrecognized BDT Network Initialization Statements

Type

Customization (optional).

General Description

This exit routine allows an installation to identify and process statements not recognized by BDTINCD. The following statements are identified and processed by BDTINCD and describe the BDT network:

SNABUF
SYSID
BDTNODE
ENDINIT

All BDT network initialization statements follow the ENDRBAM statement.

This exit routine runs only during a cold or warm start when the BDT initialization stream is processed. The exit routine can recognize user-defined initialization statements, any keywords associated with those statements (a keyword is followed by an equal sign), and any values associated with the keyword. For example, you could code the following:

```
BDTGOOP,N=SYS01
```

where:

- BDTGOOP is the initialization statement
- N is the keyword
- SYS01 is the value associated with the keyword.

BDTINCD invokes BDTUX03 for three reasons. A reason code in register 0 informs the user exit routine of the purpose for which it is run.

Reason=0 BDTINCD encounters an unrecognized initialization statement

This invocation allows the user to identify and process initialization statements not recognized by BDT. BDTINCD calls the exit routine to examine the statement. If the exit routine does not recognize it, BDTINCD processes the statement as an error.

Reason=4 BDTINCD passes a keyword and its parameters

This invocation allows the user to recognize and process keywords on a user-defined initialization statement. BDTINCD calls BDTUX03 with reason=4 as long as there is a keyword on a user-defined initialization statement.

Reason=8 EOD condition on a user-defined statement

This invocation communicates the end-of-data condition on the user-defined initialization statement. There are no more keywords to process.

See Appendix E, “Initialization Flow and User Exit Routines,” on page 169 for further information on where this exit routine occurs during initialization processing.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX03 user exit routine. Registers 2-14 are saved by BDT ASAVE processing.

Register 0
Contains a reason code:

Reason 0
Unrecognized initialization statement

Reason 4
Keyword on the user-defined initialization statement

Reason 8
End-of-data condition

Register 1
Contains the address of a parameter list. The contents of the parameter list depend upon the reason for which the exit routine runs.

Reason 0
Register 1 contains the address of a one-word parameter list:

Word 1
Address of the unrecognized initialization statement

Reason 4
Register 1 contains the address of a three-word parameter list:

Word 1
Address of the keyword on the user-defined statement

Word 2
Address of the keyword’s parameter value

Word 3
Address of a halfword field containing the length of the parameter

Reason 8
Parameters are not passed at the end of data.

Register 11
Contains the address of the initialization data CSECT, BDTINDT.

Register 12
Address of the BDT TVT.

Register 13
Points to the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point in BDTGRSV.

Register 15
Contains the entry point address into the BDTUX03 user exit routine.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX03 user exit routine, registers 2-14 of BDTINCD are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point in BDTGRSV.

Register 15
Used for delivering a return code value set by the BDTUX03 user exit routine.

RC 0
Indicates that the BDTUX03 user exit routine recognizes the statement or keyword as user-defined. Processing continues.

RC 4
Indicates that the BDTUX03 user exit routine failed to recognize the statement as user-defined. BDTINCD issues an error message, then continues processing the rest of the initialization statements.

RC 8
Indicates that the BDTUX03 user exit routine failed to recognize the statement as user-defined, or a processing error occurred that is serious enough to prevent BDT from completing initialization. BDT initialization processing terminates.

Operation

BDTINCD invokes BDTUX03 for three different reasons. It runs with a reason code of 0 when BDTINCD encounters an unrecognized initialization statement. If the exit routine sets a return code of 0 to indicate that the statement is recognized, BDTINCD scans for keywords associated with the statement. When a keyword is found, BDTUX03 is given control with reason code=4. BDTUX03 runs with reason code=4 as long as keywords are found and the exit routine sets a return code of 0. When all keywords are processed, BDTUX03 runs with a reason code of 8. The exit routine is advised in this way of an end-of-data condition.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection in your exit routine in the event of system failure. If there is no recovery in the routine, an ESTAE exit routine in BDTABMN provides clean-up in case of an abend.

Environment

Point Where Exit Routine Receives Control: This exit routine receives control during BDT cold or warm starts when the initialization files must be opened, read, and closed.

Address Space in Which Exit Routine Runs: BDT address space.

Task Block Under Which Exit Routine Runs: The exit routine runs under BDTINIT's TCB.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDINT to map the initialization data CSECT
- BDTDREG to map the registers (required by BDTXRTRN)
- BDTDTVT to map the TVT (required by BDTXRTRN)
- BDTDGSD to map the GSD (required for coding an ESTAE)

Executable Macros

- BDTXASRV to invoke abend services during abend recovery processing (required for coding an ESTAE)

What If BDTUX03 Is Not Used?

If the exit routine is not provided, BDT issues message BDT3243 (which displays the unrecognized statement), continues BDTINCD processing by reading subsequent initialization statements, and terminates at the end of initialization processing.

BDTUX04—Unrecognized Keywords on BDTNODE Statements for File-to-File Nodes

Type

Customization (optional).

General Description

This exit routine allows an installation to define and process unrecognized parameters on BDTNODE statements that define file-to-file nodes. BDTINR1 invokes this exit routine.

The BDTNODE initialization statement is used to identify all the nodes with which an installation can transfer data. A BDTNODE initialization statement also defines a node to itself. The BDTNODE statement creates the control blocks necessary for communicating with all the nodes in a network.

On the BDTNODE statement for file-to-file nodes, attributes of that BDT system can be defined, including global-local relationships, passwords used for establishing sessions, node name, and virtual logical units (VLUs) and how they are fenced.

BDTINR1, the module from which this exit routine runs, reads the BDTNODE initialization statements and builds intermediate control blocks that later become the line control tables (LCTs) and the resident logical units tables (RLTs). BDTINR2 converts the intermediate control blocks to LCTs and RLTs.

BDTINR1 recognizes only virtual logical units assigned to the interfunction communication manager (BDTIFCM), or to the transfer of data. The user exit routine functions as the first step in allowing an installation to define a VLU other than these two types. BDTUX05 in BDTINR2 must also be coded to process information passed in the intermediate control blocks built in BDTINR1. BDTINR1 invokes the user exit routine before entering error processing for invalid keywords on BDTNODE statements.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX04 user exit routine. Registers 2-14 are saved by BDT ASAVE processing.

Register 1
Contains the address of a three-word parameter list:

Word 1
Address of the unrecognized keyword encountered on the BDTNODE statement

Word 2
Address of the unrecognized keyword value

Word 3
Address of a halfword field containing the unrecognized keyword value length

Register 11
Contains the address of the initialization data CSECT, BDTINDT.

Register 12
Contains the address of the BDT TVT.

Register 13
Points to the register save area set up by ASAVE processing.

Register 14
Address of the return point, which is saved in BDTGRSV.

Register 15
The entry point address into the BDTUX04 exit routine.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX04 user exit routine, registers 2-14 of BDTINR1 are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Address of the return point, which is saved in BDTGRSV.

Register 15
Used for delivering a return code value from the BDTUX04 user exit routine:

RC 0
Indicates that the BDTUX04 user exit routine recognizes the keyword as user-defined.

RC 4
Indicates that the BDTUX04 user exit routine encounters an error while processing the keyword, but the error is not severe enough to fail BDT initialization.

RC 8
Indicates that the BDTUX04 user exit routine encounters an error while processing the keyword that is serious enough to cause BDT to terminate initialization.

Operation

BDTINR1 invokes BDTUX04 when it encounters an unrecognized keyword on the BDTNODE statement for a file-to-file node. BDTUX04 is passed the address of the initialization work area (BDTINDT), which contains the address of the keyword and associated keyword values If the user exit routine determines that the keyword is user-defined, the return code is set to 0 (in register 15); BDTINR1 continues to scan for keywords.

If the exit routine does not recognize the keyword, it passes a nonzero return code back to BDTINR1. A return code of 4 indicates a warning; however, initialization continues. A return code of 8 indicates a severe error; BDT initialization is terminated. BDT displays the bad keyword message for both return codes 4 and 8.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your exit routine. If your user exit routine does not include an ESTAE, an ESTAE exit routine in BDTABMN performs clean-up in the event of system failure.

Environment

Point Where Exit Routine Receives Control: This exit routine receives control when an unrecognized keyword is specified on the BDTNODE statement.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under the TCB established for BDTINIT.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDINT to map the initialization data CSECT

- BDTDREG to map the registers (required by BDTXRTRN)
- BDTDTVT to map the TVT (required by BDTXRTRN)
- BDTDRLT (&ENTRY=INISH) to map the intermediate control blocks used to build the RLT
- BDTDGSD to map the GSD (required for coding an ESTAE)

Executable Macros

- BDTXASRV to invoke abend services during abend recovery processing

What If BDTUX04 Is Not Used?

If the exit routine is not provided, the error handling for the unrecognized initialization statement issues error message BDT3243. It displays the unrecognized statement. BDT initialization eventually terminates. BDTINCD processing continues by reading the next initialization statement.

BDTUX05—BDTNODE Statement Keyword Processing for File-to-File Nodes

Type

Customization (optional).

General Description

This exit routine is a companion to BDTUX04. It processes unrecognized keywords found on the BDTNODE statement by exit routine BDTUX04. It runs in BDTINR2 to process user-defined fields in the intermediate storage file built by BDTINR1.

BDTINR1 runs user exit routine BDTUX04. This exit in BDTINR2 allows an installation to finish processing user-defined keywords specified on the BDTNODE statement. Both BDTUX04 and BDTUX05 run during a BDT warm or cold start.

See Appendix E, “Initialization Flow and User Exit Routines,” on page 169 for further information on where this user exit routine occurs during initialization processing.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX05 user exit routine. Registers 2-14 are saved by BDT ASAVE processing.

Register 1
Contains the address of a two-word parameter list:

Word 1
Address of the logical unit control table (LCT) entry for data transfer

Word 2
Address of the resident logical units table (RLT) node entry

Register 11
Contains the address of the BDT initialization data CSECT, BDTINDT.

Register 12
Contains the address of the BDT TVT.

Register 13
Points to the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
contains the address of the entry point into the exit routine.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX05 user exit routine, registers 2-14 of BDTINR2 are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine), and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Used for delivering a return code set by the BDTUX05 user exit routine:

RC 0
BDTINR2 assumes the user set up a special VLU type. BDTINR2 processes the next VLU.

RC 4
BDTINR2 sets up the VLU as a normal transfer VLU type.

Operation

The exit routine is passed a pointer to a logical unit control table (LCT) entry and a pointer to the resident logical unit control table (RLT). With these two control blocks, an installation can describe a VLU other than a transfer type.

If the exit routine responds with a zero return code, BDTINR2 continues processing with the next VLU. If the exit routine responds with a nonzero code, BDTINR2 marks the VLU as a normal transfer type.

If the VLU is the first VLU on a line, it is set up as an interfunction communication manager (BDTIFCM) VLU. BDTUX05 is not invoked in this instance.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your user exit routine. If you do not include an ESTAE exit routine, an ESTAE in BDTABMN provides clean-up in the event of exit routine failure.

Environment

Point Where Exit Routine Receives Control: This exit routine receives control just before a VLU is processed as a transfer VLU.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under the TCB for BDTINIT

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDINT to map the initialization data CSECT
- BDTDLCT to map the logical units table
- BDTDRLT (&ENTRY=ALL) to map the resident logical units table
- BDTDREG to map the registers
- BDTDTVT to map the TVT
- BDTDGSD to map the GSD (necessary for coding an ESTAE)

Executable Macros

- BDTXASRV to invoke abend services during abend recovery

What If BDTUX05 Is Not Used?

If exit routine BDTUX05 is not coded, the VLU is set as a transfer VLU.

BDTUX06—BDT Post-Initialization Processing

Type

Customization (optional).

General Description

This exit routine, invoked by BDTINIT, receives control after BDT has processed all of the initialization statements. This exit routine runs during cold, warm, or hot starts. It enables an installation to examine or modify data areas before initialization completes and control is passed to the multifunction monitor (BDTGRCT). For example, this exit routine can be used to turn on the TVTJMLAV bit in TVTOPTNS of the TVT to permit submission of transactions that access the BDT job message log. (Also, refer to BDTUX16 later in this chapter for that exit routine's relationship to BDTUX06.)

No parameters are passed to the exit routine and no return codes are returned from the exit routine.

BDTUX01 can be used instead of BDTUX06. BDTUX01 is called after BDTUX06.

See Appendix E, "Initialization Flow and User Exit Routines," on page 169 for more information on when the user exit routine runs during initialization processing.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX06 user exit routine. Registers 2-14 are saved by ASAVE processing.

Register 11
Contains the address of the BDT initialization data CSECT, BDTINDT.

Register 12
Contains the address of the BDT TVT.

Register 13
Contains the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Contains the entry point address into the BDTUX06 user exit routine.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX06 user exit routine, registers 2-14 of BDTINIT are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Operation

BDTINIT invokes this exit routine just before initialization completes. There are no parameters passed to this exit routine and any user data areas to be processed must be anchored in the BDT TVT.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your user exit routine. If an ESTAE exit routine is not included in the exit routine, an ESTAE in BDTABMN provides clean-up in the event of exit routine failure.

Environment

Point Where Exit Routine Receives Control: This exit routine receives control just before initialization completes.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under the TCB for BDTINIT.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDTVT to map the TVT
- BDTDINT to map the initialization data CSECT
- BDTDREG to map the registers
- BDTDGSD to map the GSD (necessary for coding an ESTAE)

Executable Macros

- BDTXASRV to invoke abend services during abend recovery

What If BDTUX06 Is Not Used?

If no user exit routine is provided, there is no impact on processing.

BDTUX07—User-Defined Parameters on the MSGCLASS Keyword of File-to-File Transactions

Type

Customization (optional).

General Description

This exit routine processes user-defined parameters specified on the MSGCLASS keyword of file-to-file transactions. Such parameters can be keywords that stand alone or keywords with values. A stand-alone keyword is not followed by an equal sign and has no associated values. A keyword with values has an equal sign and values that follow the equal sign.

The MSGCLASS keyword allows the end user to specify alternate message destinations, or XOIDs, rather than the XOID submitting the transaction. The XOID is defined as the origin of a transaction. BDTUX07 can alter the XOID of the transaction contained in the JCT by the criteria that the MSGCLASS keyword represents. The XOID contained in the JCT is used when any function of BDT issues a message on behalf of the job. BDTUX12 can then route the message to its proper destination.

The MSGCLASS keyword is entered as part of a file-to-file transaction. Those parameters recognized by BDT include *, LOG, and NONE. The * parameter specifies that messages be returned to the transaction origin and sent to the job message log. LOG specifies that messages be sent to the job message log (JML) data set only. NONE specifies that messages be sent to the system message log only.

The language processor builds a text unit for the MSGCLASS keyword. Later, BDTGRXD encounters an unrecognized parameter during the processing of the MSGCLASS keyword and this user exit routine runs.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX07 user exit routine. Registers 2-14 are saved by ASAVE processing.

Register 1
Contains the address of a three-word parameter list:

Word 1
Contains the address of a four-word parameter list:

Word 1
Byte 3 contains the length of a keyword, as provided by BDTXSUPC.

Word 2
Contains the address of the keyword associated with word 1, as provided by BDTXSUPC.

Word 3
Byte 3 contains the length of an associated parameter, as provided by BDTXSUPC.

Word 4
Contains the address of the parameter whose length is contained in word 3. This parameter is associated with the keyword pointed to by word 2, if it exists, or with the keyword MSGCLASS if no keyword exists for word 2.

Examples:

Given:
MSGCLASS(NODE=MYNODE)

Then:
Keyword is NODE, length=4 Parameter is MYNODE, length=6

Given:
MSGCLASS(MYNODE)

Then:
There is no keyword Parameter is MYNODE, length=6

Word 2
Address of the console message buffer BDTGRXD uses as the syntax analysis work area for BDTXSUPC.

Word 3
Address of the JCT entry, which can be modified by BDTUX07 to reflect the user-defined parameter(s).

Register 12
Contains the address of the BDT TVT.

Register 13
Points to the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Contains the entry point address into the BDTUX07 exit routine.

Register Conventions at Exit

Since BDTXCALL linkage is used to establish the interface to the BDTUX07 user exit routine, registers 2-14 of BDTGRXD are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine), and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the the return point, which is saved in BDTGRSV.

Register 15
Used for delivering a return code value set by the BDTUX07 exit routine:

RC 0
The parameter is recognized and processed by the exit routine.

RC 4
The parameter is not recognized.

Operation

All MSGCLASS parameters other than *, LOG, or NONE are rejected by BDTGRXD unless recognized as user-defined in the user exit routine. If the parameter is not recognized as user-defined, a return code of 4 is sent back to the transaction driver and an error message is issued stating that an invalid MSGCLASS is specified and the transaction has failed.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your user exit routine. If an ESTAE exit routine is not included with the routine, an ESTAE in BDTGRXD provides clean-up in the event of exit routine failure.

Environment

Point Where Exit Routine Receives Control: The exit routine receives control from BDTGRXD when the MSGCLASS keyword parameters are unrecognized by BDTGRXD.

Address Space in Which Exit Routine Runs: The BDT address space.

Task under Which Exit Routine Runs: This exit routine operates under the TCB for the BDT transaction driver, BDTGRXD.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDTVT to map the BDT TVT
- BDTDJCT to map the JCT
- BDTDCNS to map the console message buffer
- BDTDREG to map the registers
- BDTDGSD to map the GSD (necessary for coding an ESTAE)

Executable Macros

- BDTXASRV to invoke abend services during abend recovery

What If BDTUX07 Is Not Used?

If an exit routine is not provided and a MSGCLASS parameter other than *, LOG, or NONE is specified, the parameter is rejected and message BDT6331 is sent to the user. This means the transaction has failed.

BDTUX08—User-Defined File-to-File Transaction Keywords

Type

Customization (optional).

General Description

This exit routine allows an installation to define additional keywords that can be specified on file-to-file transactions. BDTUX08 extends the language processor's keyword table. The extension is built through the use of the BDTDKYWD macro. This macro generates entries in the user-defined keyword table extension located in a data CSECT called UX08KYWD. The macro creates one or more entries in the CSECT. BDTDKYWD issues BDTDTUD, another macro, which creates the corresponding text unit descriptor entries located in the data CSECT UX08TUD. When the language processor reaches the end of BDT's keyword table without finding the keyword, it starts searching the user-defined keyword table located in BDTUX08.

Register Conventions at Entry

Because this exit routine is only a table and not executable code, there are no registers involved.

Register Conventions at Exit

Because this exit routine is only a table and not executable code, there are no registers involved.

Operation

The data CSECT called UX08KYWD is created as the extension to the language processor's keyword table. You should use the BDTDKYWD macro to create the keyword entries that make up this extension. An associated text unit descriptor entry may also be created. The CSECT name, UX08KYWD, is generated when the BDTDKYWD macro is used.

Note: The default, &USER=YES, must be specified when coding the BDTDKYWD macro.

Since this exit routine is part of the language processor, it must be link-edited with the language processor load module, which may be loaded into the link pack area. Loading exit routines in the link pack area requires an MVS IPL. Any exit routines loaded in the link pack area (BDTUX08, BDTUX10, BDTUX19, and BDTUX28) should be link-edited at the same time. This prevents the need for you to perform more than one MVS IPL to load those exit routines.

Environment

Point Where Exit Routine Receives Control: BDTLP invokes BDTUX08 when it reaches the end of BDT's keyword table without finding a keyword.

Address Space in Which Exit Routine Runs: User's address space (JES3, TSO, or batch) and BDT address space.

Task under Which Exit Routine Runs: Because the language processor (BDTLP) is reentrant, it can be used by more than one user at a time. This exit routine operates under the TCB of the module that invokes the services of the language processor. Those modules include IGX0034, BDTGRXD, BDTCMDV, BDTTQI, and IATBDCI, a JES3 module.

PSW State: Supervisor.

Storage Protection Key: The language processor, from which this exit routine is invoked, runs in the key of the user. This key can be the user's (key 8), JES3 (key 1), or key 0 from SVC 109.

Data Areas

Mapping Macros

- BDTDKYWD—generates entries in the user-defined keyword table. This invokes BDTDTUD.
- BDTDTUD—generates entries in the text unit descriptor table

What If BDTUX08 Is Not Used?

If this exit routine is not provided, a user-defined keyword not recognized by the language processor will result in message BDT1003 being issued and the unrecognized keyword displayed. The file-to-file transaction fails.

BDTUX10—Command Password Processing

Type

Customization (optional).

General Description

This exit routine verifies a single parameter that a user can attach to a BDT command verb. The parameter can be used as a password to raise the authorization level of the user issuing the command. For example:

```
I(password) A
```

BDT permits users to attach to the command verb a 1- to 8-character name enclosed within parentheses. This appended field can be used as a password to provide increased user authorization on a single command basis. This exit routine checks for this password field and subsequently either provides or disallows increased user authorization based on the password value, the authorization level required for the specific command, and the command type.

BDT users, operating under TSO, have a default authority of 10. BDTUX10 allows an installation to raise this authority level to 15 for users who specify the correct password. An authorization level of 15 gives BDT users the authority to issue the DUMP and RETURN commands, cancel jobs, issue TQI commands, and activate the SNA manager as well as issue commands authorized at level 10. Table 7 on page 99 shows the authorization level required to issue each BDT command.

Table 7. Authorization Levels Required to Issue BDT Commands

BDT Command	Authorization Level Required to Issue the Command
CALL (X)	10
CANCEL (C)	15 if submitted from TSO or batch; 10 if submitted from an MCS or JES3 console
DUMP	15
INQUIRY (I)	10
JES	10
MESSAGE (Z)	10
MODIFY (F)	15 if used to modify TQI, modify the BDT message handler, or cancel the BDT SNA manager; 10 in all other cases
MSG	15
RESTART (R)	15 if submitted from TSO or batch; 10 if submitted from an MCS or JES3 console
RETURN	15
SEND (T)	10
START (S)	15 if used to start TQI or if submitted from TSO or batch; 10 if not used to start TQI or if submitted from an MCS or JES3 console

Table 7. Authorization Levels Required to Issue BDT Commands (continued)	
BDT Command	**Authorization Level Required to Issue the Command**
VARY (V)	10

Register Conventions at Entry

Since this exit routine is entered by a BALR instruction, that is, without going through BDT ASAVE linkage, the user exit routine must save all language processor registers in an area obtained by GETMAIN. Those registers must be restored on return to the language processor.

Register 1
Contains the address of a three-word parameter list:

Word 1
Address of the parameter

Word 2
Address of the parameter value's executable length

Word 3
Address of the BSID

Register 13
Contains the address of the BDT register save area which is located at the beginning of the BDTLP work area.

Register 14
Contains the address of the return point in the language processor.

Register 15
Contains the entry point address into the BDTUX10 user exit routine.

Register Conventions at Exit

Since the user exit routine is entered directly on a BALR instruction, BDTUX10 must save the language processor's registers in an area obtained by GETMAIN (the user exit routine must be reentrant), and restore those registers on return to BDTLP.

Register 14
Contains the address of the return point in BDTLP.

Register 15
Used for delivering a return code value from the user exit routine:

RC 0
Indicates that the exit routine recognizes the parameter and processing continues.

RC nonzero
Indicates the parameter is unrecognized and the transaction has failed.

Operation

The exit routine is called by the language processor and executes in the address space of the module that invokes the language processor. The language processor (BDTLP) may reside in the link pack area (LPA). Since the language processor is reentrant, this exit routine must also be reentrant.

Once exit routine BDTUX10 is coded, it must be link-edited with the language processor load module. An MVS IPL loads the LPA modules. All language processor user exit routines (BDTUX08, BDTUX10, and BDTUX19) should be coded before the link-edit is performed with the language processor. This way, only an MVS IPL must be performed once.

If you use this exit routine to verify passwords attached to command verbs, the passwords should be encrypted for security. Otherwise, it is possible to look at the passwords contained in BDTUX10.

The language processor invokes BDTUX10 when it detects a single parameter following a BDT command or a command verb. If the exit routine is used to verify a parameter submitted with a command or command verb, it is possible to raise the defaulted console authority level from 10 to a number between 10 and 15. This should be set in the CONSAUTH field in the console message area. The console area is located in the variable part of the BSID.

If a single parameter following a command is incorrect, an error message informs the user that the transaction code syntax is invalid. The transaction fails.

If a command is issued with a parameter that is null, or if the parameter is incorrect, an error message informs the user that the transaction code syntax is invalid. The transaction fails.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your user exit routine. If you do not include an ESTAE with the routine, an ESTAE exit routine in IGX0034, BDTCMDV, BDTGRXD, or IATBDCI (a JES3 module) cleans up after failure of the user exit routine. The ESTAE exit routine performing clean-up is dependent upon the user at the time of failure.

Environment

Point Where Exit Routine Receives Control: This exit receives control in the language processor (BDTLP) after the console authority level is defaulted to 10.

Address Space in Which Exit Routine Runs: The user's address space.

Task under Which Exit Routine Runs: The exit routine is called by the language processor (BDTLP). The exit routine could run under the TCB for BDTBATCH, BDTTSO, BDTCMDV, BDTGRXD, or IATBDCI, all of which invoke the services of the language processor.

PSW State: Supervisor.

Storage Protection Key: Since the language processor runs in either the user's address space, the BDT address space, the SVC 109 routine, or the JES3 address space, the storage protection key is 1, 8, or 0.

Data Areas

Mapping Macros

- BDTDBSID to map the BSID
- BDTDCNS to map the console message area

Executable Macros: None

What If BDTUX10 Is Not Used?

If no exit routine is provided and the user adds a parameter on a BDT command, the command will fail. Message BDT1008 will be issued to display the parameter.

BDTUX11—Unrecognized BSID Modifier

Type

Customization (optional).

General Description

This exit routine allows an installation to identify user-defined BSIDMOD codes not recognized by BDT. You can use a user-defined BSID to transmit processing information to the BDT address space from a user address space. The BSID acts as a medium for the transfer of processing information. For example, an installation can enter a user-defined command that BDT would normally reject as invalid. You can use BDTUX19 to turn the command BSID for the user command into a user-defined BSID. BDTUX11 can then receive the BSID, process it, and then tell BDT to discard it.

The BSIDMOD codes specify the purpose and use of each BSID. The BSIDMOD represents, for example, whether the BSID is for a message, transaction, JES3 command, BDT command, or native mode. Because of functional differences between JES2 and JES3, this exit routine provides some response (for example, a return code of 4) to JES3 only, if applicable. (Specific JES3 processing requirements and BDT processing on behalf of JES3 are noted as applicable. JES2 users are not affected by such JES3-specific processing.) See *z/OS BDT Diagnosis Reference* for documentation of the fields in the BSID.

The BSIDMOD is located in the fixed portion of the BSID. It identifies the type of BSID by number. These numbers include user-defined numbers in the range 128-255.

This exit routine allows the user to process BSIDMOD code values in the user range. Exit BDTUX11 is similar in function to JES3 exit routine IATUX50. However, BDTUX11 is invoked from the BDT address space, not the JES3 address space.

Register Conventions on Entry

BDTXCALL linkage is used to establish the interface to the BDTUX11 user exit routine. Registers 2-14 are saved by ASAVE processing.

Register 1
Contains the address of a one-word parameter list:

Word 1
Address of a BSID containing a BSIDMOD code in the user-defined range

Register 12
Contains the address of the BDT TVT.

Register 13
Contains the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Contains the entry point address into the BDTUX11 user exit routine.

Register Conventions at Exit

Since the BDTXCALL linkage is used to establish the interface to the BDTUX11 user exit routine, registers 2-14 of BDTCMDV are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Used for returning a return code set by the BDTUX11 user exit routine:

RC 0
The exit routine recognizes the BSIDMOD and processes it; BDTCMDV does no further processing.

RC 4
The BSIDMOD code value is recognized; BDTCMDV can send it to JES3 by the BDT/JES3 communications interface. (This return code should not be used on a JES2 system.)

RC 8
The exit routine does not recognize the BSIDMOD value; it is invalid. BDTCMDV issues message BDT9957.

Operation

This exit routine can look at the BSIDMOD field in the BSID to determine the purpose of the BSID and, based on that purpose, how it should be routed. The codes are assigned by the user.

BSIDMOD modifier codes defining the purpose and usage of each BSID are available to the user in the range 128-255. The symbol BSIDUSE1 is equated to BSID modifier code 128.

The exit routine gets control after the BSIDMOD has been checked for all BDT defined values. The BSIDMOD code must be in the user-defined range or the exit routine will not run and an error message is issued. If the value of BSIDMOD is in the user-defined range, the exit routine runs. BDTUX11 returns a return code value to communicate the results of its processing. If the return code is 0, processing continues. If the return code is 4, the BSID is shipped to JES3. A return code of 8 signifies the BSIDMOD is invalid.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your user exit routine. If an ESTAE is not included in the exit routine, an ESTAE exit routine in BDTCMDV provides clean-up in the event of exit routine failure.

Environment

Point Where Exit Routine Receives Control: BDTCMDV checks the BSIDMOD value and passes control to the user exit routine if the BSIDMOD code is within the user range (128-255).

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under the TCB for BDTCMDV.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDBSID to map the BSID
- BDTDTVT to map the TVT
- BDTDREG to map the registers
- BDTDGSD to map the GSD (for coding an ESTAE)

Executable Macros

- BDTXASRV to invoke abend services during abend recovery processing

What If BDTUX11 Is Not Used?

If no user exit routine is provided, user-defined BSID modifiers are considered invalid and the user receives an error message.

BDTUX12—BDT Message Routing

Type

Customization (optional).

General Description

This exit routine allows an installation to process messages that are sent to the user. Processing includes the ability to inspect and modify those messages. BDTUX12 should also process the alternate XOIDs that were created in BDTUX07 for user-defined MSGCLASS parameters. You can use BDTUX12 to override standard message processing. Also, BDTUX12 can be used to override message suppression to continue to route all messages to the user, not only the message log. The exit is taken out of BDTCMDV.

For example, the exit routine can be used to process messages destined for non-BDT users in the BDT network.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX12 user exit routine. Registers 2-14 are saved by ASAVE processing.

Register 1
Contains the address of the one-word parameter list:

Word 1
Address of a BSID containing the message text for routing. The text is hex 4C bytes past the beginning of the data area (BSIDDATA) of the BSID.

Register 12
Contains the address of the BDT TVT.

Register 13
Contains the address of the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point in the ASAVE routine (BDTGRSV).

Register 15
Contains the entry point address into the BDTUX12 user exit routine.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX12 user exit routine, registers 2-14 of BDTCMDV are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Used for returning a return code value set by the BDTUX12 user exit routine:

RC 0
Indicates that standard BDT message route processing should continue.

RC 4
Indicates the processing continues without message suppression.

RC 8
Indicates that the message should be routed only to the BDT log destination consoles.

RC 12
Discard the message.

Operation

This exit routine is given control by BDTCMDV, just prior to BDT message processing. The user can inspect, modify, and route the message being processed. The exit routine indicates by a return code whether the BDT message route processing should continue or be bypassed. The BSIDMCLS field indicates message suppression if it is set to BSIDSUPP.

The routine examines BSID XOID type codes (BSIDXTYP), which define the type of user receiving the message.

The XOID type codes (BSIDXTYP) currently recognized by BDT range from 1 to 7. The XOID type code values in the range 8 to 127 are reserved. The XOID type codes from a range of 128-255 are available to the user. The symbol XOIDUSER is equated to 128, the first XOID type code in the user range.

In message processing (see Figure 30 on page 105) some messages are suppressed from further routing once they are logged. You can determine if this will happen by looking at the BSID message class field (BSIDMCLS) contained in the BSID. The BSID message class bit should be set to BSIDSUPP. Suppression of messages is helpful because it prevents the originator from becoming flooded with messages returned

from normal transaction and command processing. If the BSIDMCLS bit is set to BSIDSUPP, the message is only routed to applicable entries in the SYSLOG destination routing table. The originator never sees the message unless the exit routine indicates the message should be routed even though it is suppressed.

The BDT macro BDTXMSG routes a message BSID to BDTCMDV. Then the MSGROUTE subroutine within BDTCMDV writes the message to the BDT message log. All messages are logged by BDT. If MSGROUTE is unaffected by BDTUX12, MSGROUTE routes the message to all entries contained in the SYSLOG destination routing table and then routes the message to the originator. Once the message is logged, if BDTUX12 is enabled, subsequent message routing can be varied depending on the return code set by BDTUX12.

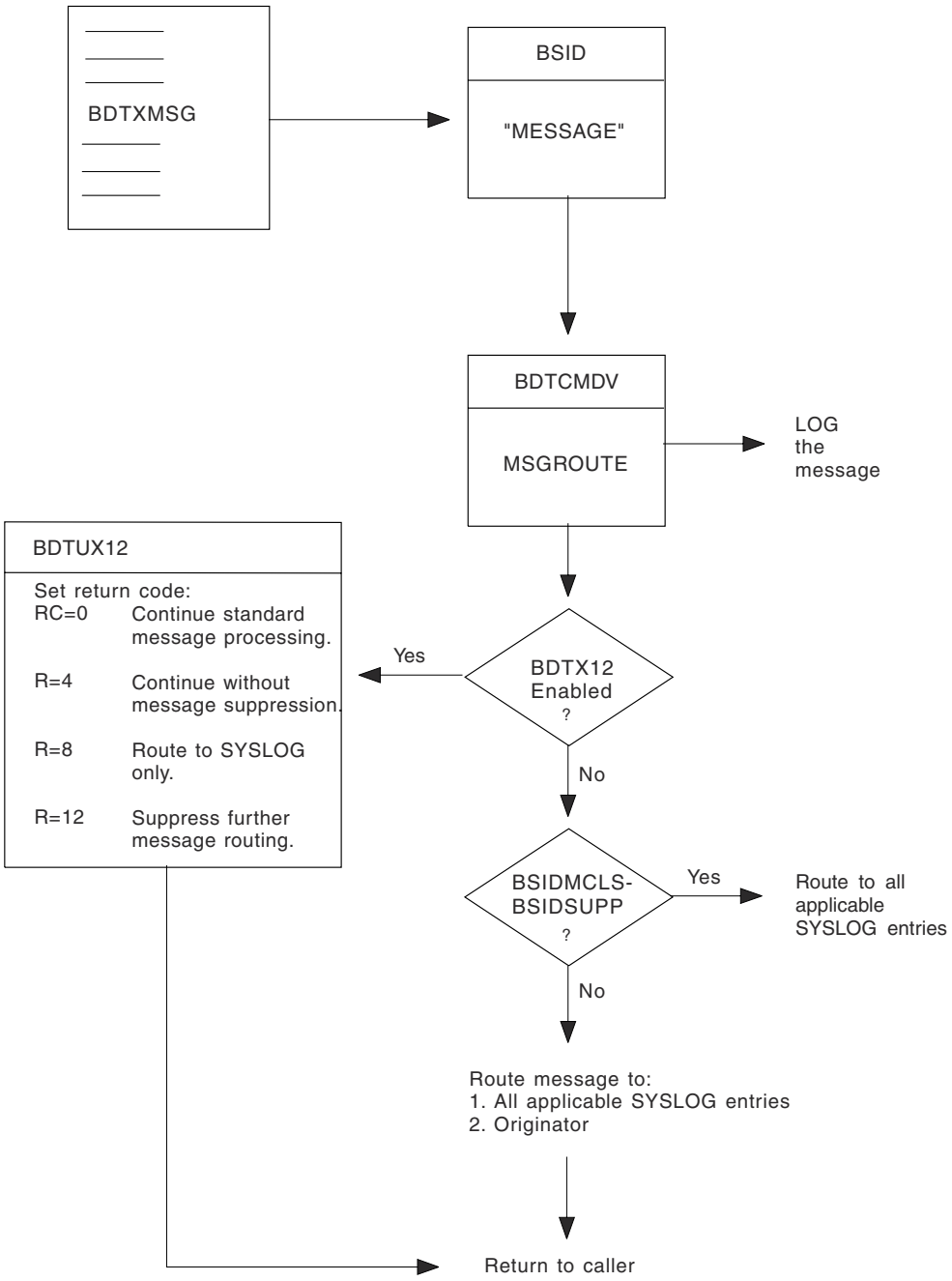


Figure 30. BDT standard message routing

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your user exit routine. If you do not include an ESTAE in the exit routine, an ESTAE exit routine established for BDTCMDV performs clean-up for the user exit routine in the event of failure.

Environment

Point Where Exit Routine Receives Control: This exit routine is invoked from BDTCMDV when processing a message.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: TCB for BDTCMDV, the BDT communications driver.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDBSID to map the BSID
- BDTDGSD to map the GSD (for coding an ESTAE)
- BDTDREG to map the registers
- BDTDTVT to map the TVT

Executable Macros

- BDTXASRV to invoke abend services during abend recovery processing

What If BDTUX12 Is Not Used?

If no exit routine is provided, standard message routing continues.

BDTUX14—BDT User-Defined XOID Type Conversion

Type

Customization (optional).

General Description

This exit routine recognizes and processes user-defined XOID type codes (XOIDXTYP) when converting from internal to external format or external to internal format. Internal to external conversion occurs when the XOID is printed in readable form as part of a message. External to internal conversion occurs on the XOID specification in a MESSAGE (Z) command and other situations. See Appendix E, “Initialization Flow and User Exit Routines,” on page 169 for a list of modules that issue the BDTXMSG macro and the types of conversions BDTXXOID performs for those modules.

A conversion takes place whenever a BDT module issues the BDTXXOID macro to request the conversion of an XOID. The conversion occurs in a subroutine located in BDTGRXD, the module from which this user exit routine runs.

See Appendix E, “Initialization Flow and User Exit Routines,” on page 169 for a diagram of internal to external conversions of XOIDs.

For internal to external conversions, you can use this exit routine to check the XOIDXTYP field in the XOID (mapped by BDTDXOID). This field must fall between 128-255 for an internal to external conversion of the XOID type which describes the type of user issuing the message.

An internal to external conversion is necessary whenever BDT wants to display the XOID part of the message text. The system identification, user type, and ddname (for TSO users, the actual user identification) are placed in the message.

For external to internal conversions, the conversion takes place when all valid BSIDXTYP codes (external format) have been tested and the type is not recognized. An example of this occurs with the MESSAGE

command. One user sends another user a message using the Z command. BDT must convert the externalized address of the message recipient into an internal format to route the message. If that user is of a type not defined to BDT but is a type that falls within the user range, the user exit routine can verify the type and convert the external XOID.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the user exit routine. Registers 2-14 are saved by ASAVE processing.

Register 0
Contains a reason code:

Reason 0
Indicates an internal to external conversion is necessary.

Reason 4
Indicates an external to internal conversion is necessary.

Register 1
Contains the address of a two-word parameter list:

Reason 0
For internal to external conversion:

Word 1
Address of the XOID data to be converted into an external representation

Word 2
Address of the external text area where the external representation of the XOID data for conversion is to be placed

Reason 4
For external to internal conversion:

Word 1
Address of the XOID data where the converted internal representation of the XOID is to be placed

Word 2
Address of the external form of the XOID to be converted into its internal XOID representation

Register 12
Contains the address of the BDT TVT.

Register 13
Contains the address of the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Entry point address into BDTUX14.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX14 user exit routine, registers 2-14 of BDTGRXD are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Used for returning a return code value set by the BDTUX14 user exit routine:

RC 0
Indicates the conversion was successful.

RC 4
Indicates the exit routine does not recognize the XOID as user defined.

Operation

BDTGRXD invokes BDTUX14 for two different situations. It is given control when the XOID, in its internal (unreadable) format, must be converted to readable format and placed into a message.

The exit routine receives control in this case if the XOIDXTYP is greater than or equal to XOIDUSER (128). The user exit routine runs with reason code 0 passed in register 0. The exit routine examines the type field of the XOID, determines the external text string that represents the internal XOID code, and moves the text string to the address passed for external XOID text. A return code of 4 informs BDTGRXD that the XOID is unrecognized and unconverted. The external XOID on the message is set to question marks.

This user exit routine also runs for external to internal conversion of user-defined XOIDXTYPs. If the external XOIDXTYP code is not recognized, BDTGRXD calls the user exit routine. If the exit routine recognizes the type, it then examines the external XOID text. It determines the internal representation of this text, then sets the type field in the passed XOIDXTYP. A return code of 4 indicates that the XOIDXTYP is unrecognized.

In either external to internal or internal to external conversions, if the conversion is successful, a return code of 0 is delivered to BDTGRXD.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your user exit routine. If you do not provide an ESTAE, clean-up for the exit routine in the event of exit routine failure is provided by an ESTAE exit routine established for the transaction driver (BDTGRXD).

Environment

Point Where Exit Routine Receives Control: This exit routine receives control in BDTGRXD whenever XOIDXTYP falls within the user range (128-255), or whenever the external format of the XOIDXTYP code is not recognized.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: TCB for the routine that issues the BDTXXOID macro (see Appendix E, “Initialization Flow and User Exit Routines,” on page 169 for a list of the modules that issue the BDTXXOID macro).

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDGSD to map the GSD (used for coding an ESTAE)
- BDTDREG to map the registers
- BDTDTVT to map the TVT
- BDTDXOID to map the XOID

Executable Macros

- BDTXASRV to invoke abend services during abend recovery processing

What If BDTUX14 Is Not Used?

No conversion is possible on user-defined types.

BDTUX15—Unrecognized Parameters on PARMS Keyword

Type

Customization (optional).

General Description

This exit routine allows installation-defined parameters on the PARMS transaction keyword to be recognized by BDTSEQ when processing a sequential file-to-file transaction. The PARMS parameter specifies information unique to BDTSEQ. (Note that PARM and USER are synonymous with PARMS.)

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the user exit routine. Registers 2-14 are saved by BDT ASAVE processing.

Register 0
Reason code as returned by BDTXSUPC:

RC 0
Indicates that a stand-alone parameter is being processed.

RC nonzero
Indicates a keyword with parameters is being processed.

Register 1
Contains the address of a four-word parameter list:

Word 1
Keyword length (zero if positional)

Word 2
Keyword address (zero if positional)

Word 3
Parameter length

Word 4
Parameter address

Register 11
Contains the address of the BDTSEQ data CSECT that is mapped by the BDTDSEQ macro. BDTDSEQ contains information about the parameter starting at label SEQSUPSC with BDTDSEQ.

Register 12
Address of the BDT TVT.

Register 13
Address of the BDT register save area set up by ASAVE processing.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Contains the entry point address into BDTUX15.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX15 user exit routine, registers 2-14 of BDTSEQ are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Used for returning a return code value from the exit routine:

RC 0
Indicates the parameter is user-defined and has been processed successfully.

RC 4
Indicates the parameter is not recognized as user-defined.

Operation

The BDTSEQ routine runs asynchronously for each side of a transaction (one side to read, the other side to write).

This exit routine runs during the parsing of the PARMS keyword. BDTSEQ passes the address of its data CSECT to the exit routine to examine unrecognized parameters on the USER keyword.

Register 11 contains the data CSECT established for the BDTSEQ routine. The user exit routine can examine the unrecognized parameter on the USER keyword, then indicate by return code whether the parameter is recognized. A 0 return code indicates that the parameter is recognized; a nonzero return code indicates that the parameter is unrecognized.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for the user exit routine. If an ESTAE is not established for the exit routine, clean-up for the user exit routine is provided by an ESTAE exit routine established for BDTSEQ.

Environment

Point Where Exit Routine Receives Control: BDTSEQ invokes the user exit routine when it encounters an unrecognized parameter on the PARMS keyword.

Address Space in Which Exit Routine Runs: BDT's address space.

Task under Which Exit Routine Runs: This exit routine runs under the TCB established for BDTSEQ.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDREG to map the registers
- BDTDTVT to map the TVT
- BDTDSEQ to map the BDTSEQ data csect
- BDTDGSD to map the GSD (for coding an ESTAE)

Executable Macros

- BDTXASRV to invoke abend services for abend recovery processing

What If BDTUX15 Is Not Used?

If BDTUX15 is not provided and a parameter unknown to BDT is entered on the PARMS keyword, message BDT4005 informs the user of an error and the file-to-file transaction fails.

BDTUX16—BDT Job Message Log

Type

Customization (optional).

General Description

BDTUX16 gives an installation access to BDT's job message log (JML) at job termination time. This can only receive control if BDTUX06 has previously set the TVTJMLAV bit on; JMLs are invalid if this bit is not on. You can request that a JML is kept for a job by specifying MSGCLASS(LOG) as part of the transaction text used to request the transfer. However, to access the JML you must use BDTUX06 to turn on the TVTJMLAV bit in TVTOPTNS of the transfer vector table (TVT). If the bit is not set on, BDT issues error message BDT6334, the JML is not accessed, and the transaction is failed.

BDTGRLG runs BDTUX16 at the end of a job, just before all JML messages are purged. BDTGRLG first runs the user exit routine to offer it access to the JML. BDT gives the exit routine a pointer to the job's JCT, and from this control block, the user exit routine can use the JCT to retrieve job information that may be needed while processing the JML.

If the user exit routine decides that the JML should not be processed, the space occupied by the JML is released. However, if the exit routine decides that the JML should be processed, subsequent BDTUX16 invocations are made, once for each message associated with a job and in the same order in which each message was written to the log. All the messages for a transaction are processed before processing another message log data set for another transaction. After the exit routine examines a message, it can tell BDT that it wants to see the next message, or it can tell BDT that it is finished looking at the set of messages associated with a job.

The exit routine is invoked a final time when BDTGRLG indicates that the JML messages have all been passed. After the final invocation of BDTUX16, BDTGRLG releases the JML space.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to BDTUX16. Registers 2-14 are saved by BDT ASAVE processing.

Register 0
Contains a reason code for BDTUX16:

Reason 0
BDTGRLG offers the exit routine access to the JML during JML close processing. The exit routine can decide at this point whether to process the JML.

Reason 4
Indicates to BDTUX16 that this is a call to process an individual message for a job. This invocation is repeated as long as there are messages in the JML for processing.

Reason 8
Indicates to BDTUX16 that the job message log has reached end of data (EOD). All messages have been retrieved and passed on to the exit routine.

Register 1
Contains the address of a one- or two-word parameter list, depending on the reason code:

Reason 0
Address of the job control table entry for this job

Reason 4
The first word contains the address of the current job message log record being processed. The second word contains the address of the job control table (JCT).

Reason 8
Address of the JCT

Register 12
Contains the address of the BDT TVT.

Register 13
Contains the address of the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point, which is stored in BDTGRSV.

Register 15
Contains the entry point address into BDTUX16.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX16 user exit routine, registers 2-14 of BDTGRLG are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine), and those registers must be restored on return to BDTGRSV by register 14.

Register 0
Used for returning a reason code from BDTUX16. This is a user-defined value that you must code. It indicates the type of error encountered if the exit routine indicates, through a return code of 8 in register 15, that an error occurred. The reason code is displayed in message BDT9901.

Register 14
Contains the address of the return point in the ASAVE routine (BDTGRSV).

Register 15
Contains the return code value delivered from the BDTUX16 user exit routine.

Reason 0
On the first invocation of the exit routine:

RC 0
Indicates that BDTGRLG should continue to call the exit routine to allow processing of the individual messages for a job.

RC 4
Indicates the BDTUX16 exit routine does not want to process the job message log and that it should be released.

Reason 4
On invocation of the exit routine for processing each individual message:

RC 0
Indicates that BDTGRLG should continue with the processing of the job message log.

RC 4
Indicates the BDTUX16 exit routine does not want to process any more records and that the job message log can be released.

RC 8
Indicates an error in the processing of the job message log.

Reason 8
At an end of data condition.

RC 0
BDTUX16 has reached end of data for the job's JML; the job message log can be released.

RC 4
An error occurred while processing the end of data condition on the job message log.

Operation

This exit routine can run for three different reasons out of BDTGRLG.

The exit routine runs the first time at the end of a job for those jobs that specified MSGCLASS(LOG). MSGCLASS(LOG) indicates that all messages associated with a job are to be accumulated in a data set and made accessible to exit routine BDTUX16 at the end of that job. Another message log data set is not processed for another transaction until the processing for these messages is complete. The exit routine is run the first time with reason=0, indicating that this is the first time the exit routine has run and that the exit routine must decide if it wants to look at the messages associated with the given job.

If the exit routine returns a 0 return code, the exit routine runs again with a reason code of 4. At this invocation it can view the first message associated with a particular job. The exit routine runs for each

message in the log as long as there are no errors in the processing of that log (indicated by a return code of 8 back to BDTGRLG). BDTGRLG continues invoking BDTUX16 with reason code 4 until the JML is exhausted.

BDTGRLG invokes the exit routine a final time when the end of data condition occurs. This indicates that all records have been taken from the job message log and passed to the user exit routine.

If an error is detected by the exit routine, BDTGRLG receives a return code of 8 from the exit routine and performs error processing.

Recovery the Exit Routine Must Establish: An ESTAE is set up for the first call to BDTUX16 by the issuer of the BDTXLOG macro. An ESTAE exit routine in BDTGRLG covers the user exit routine in subsequent calls. In the event of exit routine failure, this ESTAE exit routine releases the JML.

Environment

Point Where Exit Routine Receives Control: The exit routine is invoked when the message log data set is closed for a transaction. All the messages for a transaction are passed to the exit routine before processing another message log data set for another transaction.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: Issuer of BDTXLOG.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDGSD to map the GSD (for coding an ESTAE)
- BDTDJCT to map the JCT
- BDTDREG to map the registers
- BDTDTVT to map the TVT

Executable Macros

- BDTXASRV to invoke abend services for abend recovery processing

What If BDTUX16 Is Not Used?

If an exit routine is not provided, processing continues to purge the job message log data set. If a transaction that specifies MSGCLASS(LOG) is submitted but the TVTJMLAV bit in the TVT is not on, the transaction will fail whether or not BDTUX16 is provided.

BDTUX17—BDT Job Start

Type

Customization (optional).

General Description

This exit routine can be implemented to provide information about the execution time of a job. If your installation does not use SMF accounting, or you wish to supplement IBM's SMF provisions within BDT, this exit routine and BDTUX18 are provided for your use.

This exit routine is invoked for inbound and outbound file-to-file jobs and outbound (but not inbound) SNA NJE jobs. This exit routine runs in the scheduling processor when the processor is notified that the remote processor is ready to send or receive the job.

Register Conventions at Entry

BDTXCALL linkage is used to interface with the BDTUX17 user exit routine. Registers 2-14 are saved by BDT ASAVE processing.

Register 1
Contains the address of a one-word parameter list:

Word 1
The address of the JCT entry for the job

Register 12
Contains the address of the TVT.

Register 13
Contains the address of the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Contains the entry point address in BDTUX17.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX17 user exit routine, registers 2-14 of BDTJSFT (for file-to-file jobs) or BDTJSNT (for SNA NJE jobs) are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine), and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point in the ASAVE routine (BDTGRSV).

Operation

This exit routine runs when the scheduling processor is notified that the remote processor has scheduled the job. This exit routine can record information in the job control table about the scheduling of the DAPs while BDTUX18 can record information about the end of a transfer.

See Appendix E, “Initialization Flow and User Exit Routines,” on page 169 for a diagram that depicts the steps that precede the invocation of this user exit routine.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your user exit routine. If you do not supply an ESTAE, clean-up is provided for the user exit routine by an ESTAE exit routine established in BDTABMN.

Environment

Point Where Exit Routine Receives Control: BDTUX17 is invoked when the scheduling processor is notified that the remote processor has scheduled the job. BDTUX17 is invoked from BDTJSFT if the job is a file-to-file job, or from BDTJSNT if the job is a SNA NJE job.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under BDTGRJS’s TCB.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDGSD to map the GSD (for coding an ESTAE)
- BDTDJCT to map the JCT
- BDTDREG to map the registers
- BDTDTVT to map the TVT

Executable Macros

- BDTXASRV to invoke abend services for abend recovery processing

What If BDTUX17 Is Not Used?

If this exit routine is not coded, scheduled processing continues normally.

BDTUX18—BDT Job Termination

Type

Customization (optional).

General Description

BDTGRJS invokes this exit routine when a BDT job ends. It is intended to operate for informational purposes, not to make any modifications to the way BDT handles the job. Any return code set by the user routine is ignored.

This exit routine is invoked for inbound and outbound file-to-file jobs and outbound (but not inbound) SNA NJE jobs. This exit routine runs in the scheduling processor when the processor is notified that the remote processor has finished sending or receiving the job.

This exit routine can run from two different places in BDTGRJS, depending upon which side of the transfer completes last. See Appendix E, “Initialization Flow and User Exit Routines,” on page 169 for a diagram of the steps that lead to the invocation of this exit routine at the end of a job.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX18 user exit routine. Registers 2-14 are saved by BDT ASAVE processing.

Register 1
Contains the address of a one-word parameter list.

Word 1
Contains the address of the JCT entry for the job.

Register 12
Contains the address of the TVT.

Register 13
Contains the address of the save area set up by ASAVE processing.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Contains the entry point address in BDTUX18.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX18 user exit routine, registers 2-14 of BDTJSFT (for file-to-file jobs) or BDTJSNT (for SNA NJE jobs) are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Operation

This exit routine is given control once at the end of a job. The exit routine can be given control from two different places in BDTGRJS. The place depends upon which side (scheduling processor or remote processor) completes last.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your user exit routine in the event of failure. If an ESTAE is not included with the exit routine, clean-up for the user exit routine is provided by the ESTAE exit routine established in BDTABMN.

Environment

Point Where Exit Routine Receives Control: BDTUX18 is invoked when the scheduling processor is notified that the remote processor has finished sending the job. BDTUX17 is invoked from BDTJSFT if the job is a file-to-file job, or from BDTJSNT if the job is a SNA NJE job.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under BDTGRJS's TCB.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDGSD to map the GSD (necessary for writing an ESTAE)
- BDTDJCT to map the JCT
- BDTDREG to map the registers
- BDTDTVT to map the TVT

Executable Macros

- BDTXASRV to invoke abend services for abend recovery

What If BDTUX18 Is Not Used?

If this user exit routine is not coded, job termination proceeds normally.

BDTUX19—File-to-File Transaction Modification

Type

Customization (optional).

General Description

This exit routine allows an installation to examine a file-to-file transaction and modify it before the request is honored. It receives control before BDT accesses either data set and before RACF receives control.

The exit routine can add to or modify the BSID as well as inspect it. The exit routine runs before the total length of the BSID is placed into the header, allowing you to add or delete text units. For example, you can use BDTUX19 to place the TSO user ID of the submitter of a batch job into the BSID of a transaction.

To support the BDT-RACF interface, you can use BDTUX19 to supply or modify security information required to protect system and data set integrity.

- Security keywords and passwords can be added to BDTLP to provide security interface support for consoles.
- Security information (user IDs and passwords) not otherwise permitted to be added to the GMJD library definitions can be used.

BDTUX19 can be used to disallow password encryption for either the sending or the receiving node. To do so:

1. Locate the SECPSWD (BTUSECP) keyword text unit.
2. Store X'02' (BTURACP2) in the DATUPAR field of the text unit.

The exit routine is passed the address of the BSID and the address of the next available byte in the BSID. Since the BSID length can be altered by the exit routine, the address of the next available byte must be updated and placed into the second word of the parameter list before the exit routine returns to the language processor. You can use this exit routine to associate the TSO user ID of the submitter of a file-to-file batch (BDTBATCH) job with the transactions submitted by that job. This is accomplished by placing the user ID associated with the address space into the BSID XOID user field. Otherwise, transactions submitted through the BDTBATCH program have the batch name in the XOID. The user ID must also be placed here for later processing by BDTUX30 and BDTUX31.

BDTUX19 can determine whether a file-to-file transaction should be modified or terminated. Since this is true, the exit routine needs a method of sending an error message back to the user.

It is possible to return a message to the language processor by using the third word of the parameter list, pointed to by register 1. A system programmer can code certain conditions under which a transaction terminates because of an error. The address of an associated error message can be placed into the third word of the parameter list before returning to the language processor.

Register Conventions at Entry

Since this exit routine is entered by a BALR instruction, that is, without going through BDT ASAVE linkage, you must save the registers of the language processor in an area obtained by GETMAIN and restore those registers on return to BDTLP.

Register 1
Contains the address of a five-word parameter list:

Word 1
Address of the BSID.

Word 2
Address of the next available byte in the BSID (used if the exit routine adds any new text units to the BSID).

Word 3
Address of a variable message area to be used by BDTLP upon return from this exit routine when an error has occurred. The first byte of the message area is the length of the message area minus 1 (that is, the message area is n bytes for the message text plus 1 byte for the length of that text).

Word 4
Pointer to a flag, OPTFLAGS, that indicates if this exit routine is called from the user's address space when BDTLP first parses the transaction or from the BDT address space at merge time.

X'80'
BDT address space

Other value
User address space

Word 5
Address of the BDT TVT when OPTFLAGS=X'80'.

Word 6
Address of the MJD.

Word 7
Address of the password text unit (BTUSECP) for the FROM side of the transaction.

Word 8
Address of the password text unit (BTUSECP) for the TO side of the transaction.

Register 13
Contains the address of a register save area located at the beginning of the language processor's work area.

Register 14
Contains the address of the return point in the language processor.

Register Conventions at Exit

Since the exit routine is entered by a BALR instruction, the user exit routine must save the registers of the language processor in an area obtained by GETMAIN (since this user exit routine must be reentrant code) and restore those registers on return to the language processor.

Register 15
Used for returning a return code value set by BDTUX19:

RC 0
Indicates that the transaction modification was successful. Processing continues.

RC 4
Indicates that an error was encountered. BDTUX19 checks for a user error message pointer (third word in the parameter list pointed to by R1). The transaction fails.

Operation

This exit routine is invoked by the language processor (BDTLP) to modify the BSID before it is processed. Modification of the BSID can include the addition of text units in the MJD.

This exit routine gets control before the BSID is complete; the BSID is not always completely built until just before the transaction is ready to be processed in the BDT address space. Therefore, the exit routine is invoked before the total length is placed in the BSID header, and the exit routine may add to or modify the BSID.

Once exit routine BDTUX19 is coded, it must be link-edited with the language processor load module (BDTLP). An MVS IPL loads the exit routines into LPA. All language processor user exit routines (BDTUX08, BDTUX10, and BDTUX19) should be coded before the link-edit is performed with the language processor. This way, an MVS IPL must only be performed once.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for the user exit routine in the event of system failure. If you do not include an ESTAE in the routine, an ESTAE exit routine established by IGX00034, BDTTQI, BDTCMDV, BDTGRXD, or IATBDCI, a JES3 module, cleans up after the failure of the user exit routine. The ESTAE performing the clean-up is dependent upon the user at the time of the failure.

Environment

Point Where Exit Routine Receives Control: The exit receives control in the language processor just before the BSID is sent to BDT for processing.

Address Space in Which Exit Routine Runs: User's address space and BDT address space.

Task under Which Exit Routine Runs: The exit routine is called by the language processor (BDTLP). It can run under the TCB for IGX00034, BDTTQI, BDTCMDV, BDTGRXD, or IATBDCI, all of which invoke the services of the language processor.

PSW State: Supervisor.

Storage Protection Key: This exit routine operates in the key of the module that invokes the language processor.

Data Areas

Mapping Macros

- BDTDBSID to map the BSID
- BDTDMJD to map the MJD
- BDTDREG to map the registers
- BDTDDATU to map the BDT dynamic allocation text units in the MJD
- BDTDTVT to map the TVT

Executable Macros

- BDTXTUAM to access MJD text units. Note that BDTXTUAM can only be issued when BDTUX19 runs in the BDT address space (word 5 is nonzero and contains the TVT address). Prior to using this macro you must establish addressability to the TVT.

Programming Notes

- The language processor (BDTLP) calls BDTUX19 twice. BDTUX19 is called first in the user address space and again in the BDT address space (merge mode).
- If a transaction is self-defining (that is, it begins with "Q") and the transaction contains security keyword information (SECPSWD), word 7 and word 8 will contain the address of the SECPSWD text units when BDTUX19 is called in the user address space.

  To determine if a transaction begins with "Q", look for a "Q" in the MJD MJDXCODE field.

  To search for text units other than SECPSWD, modify the sample subroutine provided in Appendix G. The sample routine searches for specific text units while BDTUX19 is running in the user address space.
- If a transaction begins with "Q" and contains password text units, password encryption occurs while BDTLP is running in the user address space. If you try to bypass password encryption while BDTLP is running in the BDT address space by storing X'02' (BTURACP) in the DATUPAR field of the text unit, encryption will still occur.
- If the transaction is a member of a GMJD library, word 7 and word 8 contain zeroes when BDTUX19 is called in the user address space. If security keywords are present in the transaction, they are passed as parameters in word 7 and word 8 when BDTUX19 is called in the BDT address space.
- If SECPSWD (*) has been specified in the user's file-to-file transaction to request that BDTLP use userid, groupid, and password defaults, the security passwords are always encrypted. The following rules apply to groupid, userid, and password defaults:

Transaction Type
Defaults

GMJD
No defaults are provided by the security system. You must provide defaults in BDTUX19.

Batch
The defaults are the userid and password from the job card of the batch job.

TSO
The defaults are the userid and password of the TSO terminal user.

What If BDTUX19 Is Not Used?

If no exit routine is provided, the end of transaction text processing continues.

BDTUX24—Monitoring and Modifying the Type 59 SMF Record

Type

Customization (optional).

General Description

After data transfer completes, BDT writes one type 59 SMF record at the scheduling node. Exit BDTUX24 allows this SMF record to be monitored and modified before it is written. The exit routine can also suppress the writing of the SMF record.

For example, this exit routine can be used to allow an installation to write to an installation-specified data set for auditing data transfers. User exit routine BDTUX01 should be used to manage user data sets. User exit routine BDTUX01 should be used for managing the installation of user data sets.

An installation can examine or change data in the record, store information in user fields, write the data to an installation-defined (non-BDT defined) data set, or choose not to write the SMF record.

If the SMF record is written (exit routine BDTUX24 sends a return code of 0 to BDTACMN to indicate that it should be written), the SMF user exit routine IEFU83 also runs. IEFU83 allows the user to modify or suppress the type 59 record. If the return code is 4 (to indicate that the type 59 record should not be written), the storage for the SMF record is freed.

See the *z/OS MVS System Management Facilities (SMF)* manual for information on SMF user exit routine IEFU38 and the type 59 record.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the user exit routine. Registers 2-14 are saved by ASAVE processing.

Register 1
Address of the parameter list:

Word 1
Contains the address of a the SMF record.

Word 2
Contains the address of the MJD.

Register 12
Address of the TVT.

Register 13
Address of the BDT register save area.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX24 user exit routine, registers 2-14 of BDTACMN are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTACMN by register 14.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Contains a return code for BDTACMN:

RC 0
Indicates that the SMF type 59 record should be written.

RC 4
Indicates that the SMF type 59 record should not be written.

Operation

BDTUX24 is provided for those installations that want to monitor or suppress the SMF type 59 record before it is written.

For example, an installation may want to write accounting information to an installation-defined data set. While BDT can format an SMF type 59 record from the MJD and other control blocks, the system management facility user exit routine IEFU83 cannot access an installation-defined data set to write that record out.

BDTUX24 gets control from BDTACMN, the BDT accounting manager which is loaded by BDTACDV, the BDT accounting driver. The exit routine must use BDTDSMF to map the SMF record. Register 1 points to a parameter list containing the address of the SMF record. See *z/OS MVS System Management Facilities (SMF)* manual for a description of that macro and the mapping of the type 59 record.

An installation may define how it uses SMF through the SYS1.PARMLIB member SMFPRMxx. The IBM-supplied member SMFPRM00 specifies the writing of all SMF records. If TYPE is not specified in the SYS1.PARMLIB member, the default is to write all records. If the installation has its own SYS1.PARMLIB members SMFPRMxx, these members may have to be changed to include or suppress type 59 records.

If an installation has supplied an IEFU83 exit routine, it might have to change the exit routine to select or suppress the type 59 record.

Recovery the Exit Routine Must Establish: This exit routine runs under the ESTAE exit routine established in BDTACDV, the BDT accounting manager.

Environment

Point Where Exit Routine Receives Control: This exit routine receives control from BDTACMN, just prior to the writing of the SMF type 59 record.

Information on Entry: This exit routine is passed the address of the SMF record and the address of the MJD.

The record contains two 40-byte fields for users at SMF59US1 and SMF59US2.

The exit routine also has access to the TVT.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: BDTACDV.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDGSD to map the GSD (for coding an ESTAE exit routine)
- BDTDREG to map the registers
- BDTDTVT to map the TVT
- BDTDSMF is the mapping macro for the SMF type 59 record.

Executable Macros

- BDTXASRV to invoke abend services for abend recovery processing

What If BDTUX24 Is Not Used?

If no exit routine is provided, BDTACMN writes the SMF record.

BDTUX25—Entry Level Authorization in the BDT Address Space

Type

Authorization (required).

General Description

This exit routine is invoked from module BDTCMDV for all file-to-file transactions when TQI is disabled, and for all BDT and JES3 commands whether TQI is enabled or disabled.

Note: If TQI is enabled for file-to-file transactions, this exit routine is never entered for the transaction. User exit routine BDTUX29, located in BDTTQI, must be coded to check the authorization of transactions issued when TQI is enabled.

The BSID is available for inspection before the command or transaction is sent on for processing.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX25 user exit routine. Registers 2-14 are saved by BDT ASAVE processing.

Register 1
Address of a two-word parameter list:

Word 1
Address of the BSID.

Word 2
Address of the variable portion of the BSID:

- For commands – address of the CONSAREA
- For file-to-file transactions – address of the MJD.

Register 12
Address of the TVT.

Register 13
Points to the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point in the ASAVE routine (BDTGRSV).

Register 15
Contains the entry point address into the BDTUX25 user exit routine.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX25 user exit routine, registers 2-14 of BDTCMDV are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point in the ASAVE routine (BDTGRSV).

Register 15
Used to pass a return code back to BDTCMDV:

RC 0
Authorization verified.

RC 4
The command or file-to-file transaction should be canceled; an error message is issued indicating that the command or transaction failed an authorization check and the command or transaction is failed.

Operation

This exit routine allows an installation to examine the entire BSID. Fields of particular interest include the BSIDXOID, which is the origin of the user, and BSIDMOD, which indicates whether the communication with BDT is a JES3 or BDT command, or a file-to-file transaction. The variable portion of the MJD, which is built for a transaction by the language processor, is not yet merged for GMJD transactions. Q-type transactions are complete at this point, however. The user exit routine can examine the text units built in the MJD.

If a nonzero return code is returned by the exit routine, the file-to-file transaction or command is canceled and a message is sent to the security console, the BDT log, and the user. The message states that the command or transaction failed the authorization check and sends a return code back to the user.

If an abend occurs in BDTUX25, a message is sent stating the name of this exit routine with an abend code, notifying the user that the command or transaction failed. The message is viewed by the user and the operator, and is recorded on the log.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for the user exit routine in the event of system failure. If an ESTAE is not included as part of your user exit routine, clean-up for the routine is provided by an ESTAE exit routine established in BDTCMDV.

Environment

Point Where Exit Routine Receives Control: The user exit routine receives control in BDTCMDV.

Address Space in Which Exit Routine Runs: The exit routine executes as part of the BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under the TCB of BDTCMDV, a resident module and subtask of BDT.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDBSID to map the BSID
- BDTDCNS (with parameters &TYPE=CMA,ICMB,OCMB) to map the command buffer in the variable portion of the BSID.
- BDTDGSD to map the GSD (for coding an ESTAE exit routine)

- BDTDMJD to map the master job definition
- BDTDREG to map the registers
- BDTDTVT to map the TVT

Executable Macros

- BDTXASRV to invoke abend services for abend recovery processing

What If BDTUX25 Is Not Used?

This exit routine must be coded before BDT will allow file-to-file transactions to be processed when TQI is disabled, and commands to be processed regardless of TQI's condition. If this exit routine is not coded, the transactions and commands are canceled.

BDTUX26—Global Node Level Authorization

Type

Authorization (required).

General Description

This is a BDT authorization exit routine invoked from the transaction driver, BDTGRXD, which runs for every file-to-file transaction. This exit routine runs in BDTGRXD in the global node after the MJD is complete and before BDTGRXD places the BDT job on the work queue.

The exit routine can look at the MJD, which contains a fixed and variable section. The fixed section describes all the information necessary to get the job through the system. The variable section contains the text units for the dynamic allocation of data sets to make the data transfer. It also contains BDT text units which describe the particular kind of processing to be performed for the transaction. These text units have nothing to do with dynamic allocation but simply use the format of MVS text units to pass information through the BDT system.

The exit routine can also modify the JCT or the MJD.

BDTUX26 runs prior to writing the MJD and JCT to the work queue. Once this is completed, the "sending" BSID is released. BDTUX26 allows the user to modify the MJD and JCT before writing each to DASD.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX26 user exit routine. Registers 2-14 are saved by BDT ASAVE processing.

Register 1
Address of the parameter list:

Word 1
Address of the MJD.

Word 2
Address of the JCT.

Register 13
Points to the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Points to the entry point address into the BDTUX26 user exit routine.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX26 user exit routine, registers 2-14 of BDTGRXD are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within the user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point in the ASAVE routine (BDTGRSV).

Register 15
Used to pass a return code back to BDTGRXD:

RC 0
Authorization granted.

RC 4
Authorization failed; an error message is issued indicating that the transaction failed an authorization check, and the transaction is failed.

Operation

The global authorization exit routine should provide network-wide checks on file-to-file transactions entering BDT. It can verify or deny the use of any data sets that are protected (such as SYS1. data sets). An installation might decide to cancel any file-to-file transactions from TSO users with a certain prefix identification. This is the place to provide a high-level check to catch potential violations of system security. You can avoid needless overhead if the transaction is disapproved in this exit routine, rather than immediately before the job is to be executed (see BDTUX27).

This exit routine is passed the address of the MJD and JCT as parameters. The MJD contains a fixed and variable portion, both of which can be inspected or modified by this exit routine.

The MJD's fixed portion contains the job name, priority, source, and destination locations, and the transaction origin identification which includes the type and actual identification of the user requesting the transaction. The variable portion of the MJD contains the text units necessary to perform dynamic allocation of a data set and carries processing information relevant to BDT.

If the exit routine passes a return code of 4, the transaction is canceled and a message is sent to the security console, the log and the end user. The message states that the command failed the authorization check and sends a return code back to the user.

If an abend occurs in an authorization exit routine, a message is sent identifying the name of this exit routine with an abend code, notifying the user that the requested command or transaction is terminated. The message is sent to the user and the operator, and is recorded on the log.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for the user exit routine in the event of system failure. If an ESTAE is not included as part of your user exit routine, clean up for the routine is provided by an ESTAE exit routine established for BDTGRXD.

Environment

Point Where Exit Routine Receives Control: BDTGRXD invokes BDTUX26 after the MJD for a file-to-file transaction is complete and before BDT places the job on the work queue.

Address Space: The exit routine executes as part of the BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under BDTGRXD's TCB.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDGSD to map the GSD
- BDTDJCT to map the JCT
- BDTDMJD to map the MJD
- BDTDREG to map the registers
- BDTDTVT to map the TVT

Executable Macros

- BDTXASRV to invoke abend services during abend recovery processing

What If BDTUX26 Is Not Used?

If the exit routine is not coded, the file-to-file transaction is canceled. This exit routine must be coded before BDT can process file-to-file transactions.

BDTUX27—Node Level Authorization

Type

Authorization (required).

General Description

This is a BDT authorization exit routine from BDTGRJR that runs for every file-to-file transaction. The exit routine runs on both sides of a data transfer just before the DAPs are scheduled for a data transfer. The exit routine can look at the MJD. The MJD contains a fixed and variable section. The fixed section describes all the information necessary to get the job through the system. The variable section contains the text units that describe the data set information involved in a transfer of data and other information about the file-to-file transaction.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX27 user exit routine. Registers 2-14 are saved by BDT ASAVE processing.

Register 1
Points to a one-word parameter list:

Word 1
Address of the MJD.

Register 13
Points to the register save area set up by ASAVE processing.

Register 14
Points to the address of the return point, which is saved in BDTGRSV.

Register 15
Contains the entry point address into the BDTUX27 user exit routine.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX27 user exit routine, registers 2-14 of BDTGRJR are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRJR by register 14.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Used to pass a return code back to BDTGRJR:

RC 0
Authorization granted.

RC 4
Authorization failed; BDTGRJR cancels the transaction.

Operation

This exit routine allows an installation to examine the MJD.

If the exit routine passes a return code of 4, the file-to-file transaction is canceled and a message is sent to the security console, the log, and the end user. The message states that the transaction failed the authorization check and sends a return code back to the user.

If an abend occurs in an authorization exit routine, a message is sent identifying the name of this exit routine with an abend code, notifying the user that the requested command or transaction is terminated. The message is viewed by the user and the operator, and is recorded on the log.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for the user exit routine in the event of system failure. If an ESTAE is not included as part of your user exit routine, clean-up for the routine is provided by an ESTAE exit routine established in BDTABMN.

Environment

Point Where Exit Routine Receives Control: The user exit routine receives control from BDTGRJR after a new FCT is obtained by BDTGRFC and added to the FCT chain. A new task is created for this DAP, then BDTGRJR gets control.

BDTGRJR gives control to BDTUX27 before the DAP and the DAP data CSECT are loaded.

Address Space in Which Exit Routine Runs: BDTUX27 runs in the BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under the DAP's TCB.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDGSD to map the GSD
- BDTDMJD to map the MJD
- BDTDREG to map the register save area
- BDTDTVT to map the BDT TVT

Executable Macros

- BDTXASRV to invoke abend services during abend recovery processing

What If BDTUX27 Is Not Used?

If the exit routine is not coded, the file-to-file transaction is canceled. This exit routine must be coded before BDT can process file-to-file transactions.

BDTUX28—MCS Console Authorization

Type

Authorization (required).

General Description

This exit routine allows an installation to examine the attributes of an MCS console, which are set at the time of system generation. Based on those attributes, the user exit routine can be coded to assign an authorization level for BDT work requests. In later authorization exit routines, this authorization level and the work request associated with the MCS console can be examined and authorized or not authorized.

The exit routine gets control in BDTSS34, the module that handles all MCS console requests.

Table 7 on page 99 shows the authorization levels required of all BDT commands.

Note:

1. This exit routine is invoked only for BDT commands having a prefix of “bdt-char” or “BDT”. This exit routine is not invoked if the prefix is “F” (MVS MODIFY command).
2. Exit routine BDTUX25 checks all commands, no matter how they enter BDT.

Register Conventions at Entry

Because this exit routine is entered using a BALR instruction, the routine must save all registers on entry to the exit routine. Also, because BDTSS34, the routine that calls this exit routine, is reentrant, the user exit routine must also be reentrant.

Register 1
Contains the address of a parameter list:

Word 1
Address of an MCS console origin (SSCMSCID).

Word 2
Address of MCS console authorization information flags (byte 1):

- X‘80’–information only
- X‘40’–command group I-system
- X‘20’–command group II-I/O
- X‘10’–command group III-console

The fourth byte (X‘80’) indicates master console. One bit past the master console indicator is the pseudo-master console.

Word 3
Address of an authorization level set by the user exit routine (byte 1).

Register 13
Points to a register save area set up by BDTSS34.

Register 14
Contains the address of the return point in BDTSS34.

Register 15
Contains the entry point address into the BDTUX28 user exit routine.

Register Conventions at Exit

Since the user exit routine is entered directly on a BALR instruction, the routine must restore the BDTSS34 registers on return.

Register 14
Contains the address of the return point in BDTSS34.

Register 15
Used for delivering a return code value from the exit routine:

RC 0
Indicates a normal return. Processing continues.

RC 4
Indicates that an error was encountered. An error message is issued indicating that the command or transaction failed an authorization check and that the transaction failed.

Operation

This exit routine is called by BDTSS34 and executes in the address space of the user. Because BDTSS34 is reentrant code, this exit routine must also be reentrant.

BDTSS34 invokes BDTUX28 for the assignment of authorization levels to all MCS consoles. The user exit routine is passed, as part of the parameter list, the address of the MCS console authorization flags. The routine must determine, based on these flags, what authorization level to assign. This should be set in the CONSAUTH field in the console message area. See Table 7 on page 99 for the proper authorization levels. The exit routine returns the assigned authorization level in the field pointed to by word 3 of the parameter list. The third word of the parameter list should be passed on entry.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for the user exit routine in the event of system failure. If you do not include an ESTAE in the routine, an ESTAE that cleans up after the failure of the user exit routine is established by IEFJRASP. IEFJRASP acts as an interface between the communications task (COMTASK) and BDTSS34.

Environment

Point Where Exit Routine Receives Control: The exit routine receives control in BDTSS34.

Address Space: Link pack area.

Task under Which Exit Routine Runs: The exit routine is called by BDTSS34 and runs under the TCB for the user.

PSW State: Supervisor.

Storage Protection Key: Key 0.

Data Areas

Mapping Macros: BDTDREG

Executable Macros: None

What If BDTUX28 Is Not Used?

If no exit routine is provided, BDT will not process commands issued via the command character.

BDTUX29—Initial Authorization of TQI-Enabled Transactions

Type

Authorization (required).

General Description

This exit routine allows an installation to authorize file-to-file transactions upon entry to the TQI address space when TQI is enabled. If TQI is disabled, BDTUX25 is invoked to authorize file-to-file transactions, and transactions flow as commands always flow—through BDTCMDV, then BDTGRXD.

This exit routine runs in BDTTQI. BDTTQI reads the records from the TQI checkpoint data set, then passes the transaction to BDTGRXD. See "How Authorization Exit Routines Fit into the Flow in a BDT File-to-File Subsystem" on page 71 for more information on the logical flow of transactions when BDT TQI is enabled and disabled.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX29 user exit routine. Registers 2-14 are saved by BDT ASAVE processing prior to returning to BDTTQI.

Register 1
Contains the address of a two-word parameter list:

Word 1
Address of a BSID.

Word 2
Address of the MJD.

Register 12
Contains the address of the BDT TVT.

Register 13
Contains the address of the register save area set up by ASAVE processing.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX29 user exit routine, registers 2-14 of BDTTQI are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either a GETMAINed area or one in your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 15
Contains the entry point address into the BDTUX29 user exit routine.

RC 0
The exit routine authorizes the transaction for processing.

RC 4
The exit routine does not authorize the transaction: an error message is issued indicating that the transaction failed an authorization check and the transaction failed.

Operation

This exit routine allows an installation to examine the entire BSID for a file-to-file transaction to determine if that transaction should be processed. Fields in both the fixed part of the BSID, which includes information about the user, and the variable portion of the MJD can be examined. The variable portion of the MJD, which is built for a transaction by the language processor, is not yet merged for GMJD transactions. Q-type transactions are complete at this point, however. The user exit routine can examine the text units built in the MJD.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your user exit routine. If you do not provide an ESTAE, an ESTAE exit routine in BDTTQI cleans up in the event of exit routine failure.

Environment

Point Where Exit Routine Receives Control: The exit routine receives control just before it is entered on the pending table for TQI.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under the TCB for BDTTQI.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDBSID to map the BSID
- BDTDTVT to map the TVT
- BDTDREG to map the registers
- BDTDMJD to map the MJD

Executable Macros

- BDTXASRV to invoke abend services for abend recovery procedures

What If BDTUX29 Is Not Used?

If no exit routine is provided, the file-to-file transaction will fail.

BDTUX30—Dynamic Deallocation

Type

Customization (optional).

General Description

This exit routine allows your installation to specify that a transaction submitter is the owner of a specific tape or DASD data set. This exit routine receives control from the dynamic deallocation module (BDTGRDA) and contains a pointer to the MJD for the transaction for which the data set was allocated. This exit routine receives control on the “from” side (source) and on the “to” side (destination) for each transaction.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX30 user exit routine. Registers 2-14 are saved using BDT ASAVE processing in BDTGRSV.

Register 0
Contains a reason code:

Reason 0
Called on “from” side for normal termination.

Reason 4
Called on “from” side for abnormal termination, restart.

Reason 8
Called on “from” side for abnormal termination, no restart.

Reason 12
Called on “to” side for normal termination.

Reason 16
Called on “to” side for abnormal termination, restart.

Reason 20
Called on “to” side for abnormal termination, no restart.

Register 1
Contains the address of a parameter list.

Word 1
Address of the MJD.

Word 2
Address of the ddname.

Word 3
Data set information; defaults are used if a parameter is not specified.

- Status (byte 1)
  - X‘01’–OLD
  - X‘02’–MOD
  - X‘04’–NEW
  - X‘08’–SHR
- Disposition (byte 2)
  - X‘01’–UNCATLG
  - X‘02’–CATLG
  - X‘04’–DELETE
  - X‘08’–KEEP

Reason Code
Disposition Used

0, 12
Normal disposition (the first DISP parameter from the transaction or its default)

4, 16
KEEP (X‘08’)

8, 20
Conditional disposition (the second DISP parameter from the transaction or its default)

Word 4
Length (hexadecimal) of user message area; set at X‘AF’ (175 bytes).

Word 5
Address of a user-provided message area to be printed by the calling module. (This field is zeroed upon entry.)

Register 12
Address of the BDT TVT.

Register 13
Pointer to the register save area set up by ASAVE processing.

Register 15
Address of the entry point in the BDTUX30 user exit routine.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX30 user exit routine, registers 2-14 of BDTGRDA are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point in the ASAVE routine (BDTGRSV).

Register 15
No return codes; the contents of register 15 are ignored.

Note: The last word of the parameter list (pointed to by register 1) contains the address of a message area that can be used to return a message.

Operation

Module BDTGRJR, which runs under the dynamic application program's (DAP's) TCB, calls the dynamic allocation/deallocation module (BDTGRDA); it is from BDTGRDA that this exit routine receives control. Both modules, BDTGRJR and BDTGRDA, reside in BDTNUC, the BDT nucleus. Furthermore, because BDTGRDA is reentrant, this exit routine must also be reentrant.

The exit routine receives control with pointers to the MJD of the transaction and the ddname. The MJD contains the XOID with the origin information concerning the transaction. The user ID of the submitter of the transaction is taken from the TSO user ID field (MJDUSID) or the user fields (MJDXRU1 and MJDXRU2). The type of user who submitted the transaction is determined from the MJDXTYP field. The ddname can be used to issue a RDJFCB and then a DEVTYPE macro to determine the volume serial numbers and other device characteristics associated with a data set.

To return a message which is printed by BDT, the exit routine must place the length of the message in the first byte of the message area followed by the message text. The address of this message is contained in the parameter list pointer to by register 1. This message area can be filled in by the exit routine. This message must not exceed 174 bytes of text; if you do exceed this limit, BDT will truncate the excess portion of the text without returning a message to that effect. If, on return from the exit routine, BDTGRDA determines that the first byte of the message area is nonzero, BDTGRDA prints the message. No messages are printed following an abnormal termination of the exit routine.

You can also use BDTUX19 to associate a user ID with a transaction. When a transaction is submitted from a batch job, there is no user ID in the XOID; the XOID contains the batch job name, only. However, BDTUX19 can be used to add the address of the user to the 8-byte user field in the XOID.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your exit routine in the event of an abend. If there is no recovery in the routine, the ESTAE recovery routine established by BDTGRDA gains control.

Environment

Point Where Exit Routine Receives Control: This exit routine receives control from the dynamic deallocation module (BDTGRDA) called by BDTGRJR.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under the TCB of the scheduled DAP (BDTSEQ or BDTPDS).

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDMJD to map the MJD to obtain access to the XOID and the account (ACCT) parameter. (The MJD is **not** updated with transaction statistics when this exit routine gains control.)
- BDTDGSD to map the GSD (if you code an ESTAE)
- BDTDREG to map the registers
- BDTDTVT to map the BDT TVT

Executable Macros

- BDTXASRV to invoke abend services (if you code an ESTAE)
- BDTXTUAM to retrieve text units in the MJD

Data Management Macros

- DEVTYPE to request information on the device characteristics associated with a specific DD statement

  Note: Standard data management macros are required to map the JFCB (IEFJFCBN) returned from by RDJFCB and the data returned by the DEVTYPE macro (IHADVA).

What If BDTUX30 Is Not Used?

If this exit routine is not provided, your installation will not be able to associate a data set with the submitter of a transaction. BDT can appear (depending upon your use of OPEN, CLOSE, and EOV exits) as owner of all data sets created by BDT transactions.

BDTUX31—INQUIRY and MODIFY Command Authorization

Type

Authorization (required).

General Description

This exit routine allows your installation to validate and authorize a user to inquire about or modify a transaction. This is actually a second level of authorization. The command must have been previously authorized by one of the other BDT-required authorization exit routines. This exit routine provides further authorization refinement by checking for user/owner authorization to inquire about or modify a specific transaction. Thus, you can guarantee only the original user or an authorized user (such as the operator) can change or inquire about transactions. This exit routine allows you to compare the user ID for the command to the user ID of the transaction to be processed. After comparing the two user IDs or verifying that the command was submitted by an authorized user, you can decide (based on this comparison) whether to process all transactions associated with this command, to process this transaction and validate the next transaction, to reject this transaction and check the next, or reject the entire command.

BDTUX31 is invoked immediately before a command is processed in response to either an INQUIRY (I) or MODIFY (F) command. The exit routine is entered once for each transaction processed; this then, could be numerous times for a single command if more than one transaction on the work queue is associated with that command. It is your responsibility to control the number of messages returned. For example, on the first entry for this exit routine you can cause BDT to return a message that indicates that only transactions belonging to the command submitter will be processed.

Reason code 252 is returned to the calling routine following the exit call for the last transaction associated with a specific command. The exit routine is entered with reason code 252 only after all return codes previously returned were either 4 or 8 (that is, reenter the exit routine for the next transaction associated with this command) but not for return codes of either 0 or 12 (that is, acceptance of rejection of all transactions associated with this command). This final call occurs because there are no more transactions associated with this command to check or process.

This exit routine processes only INQUIRY and MODIFY commands that are transaction specific. BDTUX31 is not entered, for example, for an I, B (Inquiry on Backlog) command, because this command requests summary information rather than transaction-specific information.

This exit routine is called from a common routine in BDTIQDV which is called from one of the following four modules, depending on the specific INQUIRY or MODIFY command:

Module
Command

BDTIQQU
I,J= I,P= I,Q F,J=

BDTIQAC
I,A

BDTIQDS
I,DSN

BDTDJIM
All DTC commands (including I,NET;I NET ID=; and F,NET)

BDTUX31 is invoked for the following reasons; the reason code (passed in register 0) informs the exit routine of the purpose for which it is called. A valid return code must be returned for each entry except a last entry (reason code 252) when the return code is ignored.

Reason Code
Explanation

0
Exit routine is called by a common routine in BDTIQDV which was called from BDTIQAC for I,A commands. This invocation validates an Inquiry Active command prior to display of a transaction.

4
Exit routine is called by a common routine in BDTIQDV which was called from BDTIQDS for I,DSN commands. This invocation validates an Inquiry by Data Set Name command prior to display of a transaction.

8
Exit routine is called by a common routine in BDTIQDV which was called from BDTIQQU for I,J= commands. This invocation validates the Inquiry by Job Name or Number command prior to display of a transaction.

12
Exit routine is called by a common routine in BDTIQDV which was called from BDTIQQU for I,P= commands. This invocation validates the Inquiry by Priority command prior to display of a transaction.

16
Exit routine is called by a common routine in BDTIQDV which was called from BDTIQQU for I,Q commands. This invocation validates the Inquiry on Work Queue command prior to display of a transaction.

20
Exit routine is called by a common routine in BDTIQDV which was called from BDTDJIM for I,NET commands. This invocation validates the Inquiry on a Network command prior to display of a transaction.

24-124
Reserved.

128
Exit routine is called by a common routine in BDTIQDV which was called from BDTIQQU for F,J= commands. This invocation validates the Modify-Transaction command prior to changing the status of a transaction.

132
Exit routine is called by a common routine in BDTIQDV which was called from BDTDJIM for F,NET commands. This invocation validates the Modify-Network Transaction command prior to changing the status of a transaction in a network.

136-248
Reserved.

252
Exit routine is called once more following the exit call for the last entry associated with a specific command to provide an end-of-command processing notification to the user.

Register Conventions at Entry

BDTXCALL linkage is used to establish the interface to the BDTUX31 user exit routine. Registers 2-14 are saved by BDT ASAVE processing in BDTGRSV.

Register 0
Contains a reason code:

Reason 0
Validate I,A commands.

Reason 4
Validate I,DSN commands.

Reason 8
Validate I,J= commands.

Reason 12
Validate I,P= commands.

Reason 16
Validate I,Q commands.

Reason 20
Validate I NET commands.

Reason 128
Validate F,J= commands.

Reason 132
Validate F,NET commands.

Reason 252
Last entry for this command.

Register 1
Contains the address of the parameter list.

Word 1
Address of the XOID of the command.

Word 2
Address of the XOID of the transaction.

Word 3
Length of the command text.

Word 4
Address of the command text.

Word 5
Address of the JCT for reason codes 8, 12, 16, 20, 128, 132.

Word 6
Length of message area (X'AF').

Word 7
Address of a user message area (this area is zeroed on each entry to the exit routine).

Word 8
Address of a 1-word user area (this area is zeroed on the first entry for each command). This area can be used to save the original reason code for a last (reason code 252) entry and to set a flag to indicate if any transactions have been processed for a command.

Register 12
Address of the BDT TVT.

Register 13
Pointer to the register save area set up by ASAVE processing.

Register 14
Address of the return point in the ASAVE routine (BDTGRSV).

Register 15
Address of the entry point in the BDTUX31 user exit routine.

Register Conventions at Exit

Because BDTXCALL linkage is used to establish the interface to the BDTUX31 user exit routine, registers 2-14 of BDTINIC are saved in BDTGRSV. You must store registers 12, 13, and 14 in an area you provide (either an area obtained by GETMAIN or one within your user exit routine) and those registers must be restored on return to BDTGRSV by register 14.

Register 14
Contains the address of the return point, which is saved in BDTGRSV.

Register 15
Used for returning a return code value set by the BDTUX31 user exit routine.

For return codes 0, 4, 8, and 12.

RC 0
Indicates that this command and all transaction checking associated with this command are accepted. Do not reenter the exit routine for this command.

RC 4
Indicates that this transaction is accepted and the exit routine should be reentered to check the next transaction associated with this command.

RC 8
Indicates that this transaction is rejected and the exit routine should be reentered to check the next transaction associated with this command.

RC 12
Indicates that this command is rejected as is all transaction checking associated with it. Do not reenter the exit routine for this command.

Note: For all reason codes except 252, a valid return code must be returned. If a valid return code is not returned, the command is rejected, and a user message is not printed. For entry reason code 252, the contents of register 15 are not checked.

Operation

When a user issues an INQUIRY (I) or MODIFY (F) command, this exit routine should verify that users are only displaying or modifying their own transactions or those transactions which that user is authorized to display or modify. Therefore, this exit routine should be used to prevent users from either displaying or modifying all transactions unless they are authorized.

The input passed to the exit routine includes a reason code indicating the type of command issued (INQUIRY or MODIFY). Furthermore, the parameter list contains the address of the XOID of the transaction and the address of the XOID of the command. Origin information in the XOIDs can be compared to determine whether the transaction should be processed. The exit routine returns a code that indicates whether to process the transaction and whether to continue processing other transactions associated with this command. The user may also return the address of a message (maximum of 174 bytes) to be printed; this message is returned in the message area whose address is contained in the parameter list pointed to by register 1.

This exit resides in reusable code; therefore, the exit routine must also be reusable.

Recovery the Exit Routine Must Establish: You should provide ESTAE protection for your exit routine in the event of an abend. If there is no recovery in the routine, the ESTAE recovery routine established by BDTIQDV gains control. This ESTAE rejects the command and prints a message.

Environment

Point Where Exit Routine Receives Control: This exit routine receives control from a subroutine in BDTIQDV called by BDTIQQU, BDTIQAC, BDTIQDS, or BDTDJIM.

Address Space in Which Exit Routine Runs: BDT address space.

Task under Which Exit Routine Runs: This exit routine runs under the TCB of BDTIQDV.

PSW State: Supervisor.

Storage Protection Key: BDTKEY (key 8).

Data Areas

Mapping Macros

- BDTDXOID to map the XOID for both the command and transaction
- BDTDGSD to map the GSD (if you code an ESTAE)
- BDTDJCT to map the JCT
- BDTDREG to map the registers
- BDTDTVT to map the TVT

Executable Macros

- BDTXASRV to invoke abend services (if you code an ESTAE)

What If BDTUX31 Is Not Used?

If this exit routine is not provided, BDT will reject the INQUIRY and MODIFY commands associated with a specific transaction.

Chapter 11. Mapping Macro Reference

This chapter describes the BDT mapping macros that you can use when coding user exit routines. The macros are listed in alphabetic order.

The mapping macros identified in this chapter are provided as Product-Sensitive programming interfaces for customers by BDT.

Attention: Do not use as programming interfaces any BDT mapping macros other than those identified in this chapter.

BDTDBSID

BDTDBSID maps a BDT subsystem interface data area (BSID).

```
BDTDBSID
```

BDTDCNS

BDTDCNS maps five console buffer areas, including the console message area, the input console message buffer, the output console message buffer, the input console message action codes, and the intercom request buffer.

```
BDTDCNS           {CMA   }
                  {INPUT }
           &TYPE= {ICMB  }       one or list of
                  {FCTQ  }
                  {OCMB  }
                  {OUTPUT}


            &CODES= {YES}
                    {NO }


                     {CONS}
            &PREFIX= {CON }
                     {CN  }
```

Parameter	Subparameter	Explanation
&TYPE=		Specifies the type of console buffer area to be mapped.
	CMA	Specifies that the console message area, used to process input commands, is to be mapped.
	INPUT	Specifies that a console message area, used to process input commands, is to be mapped.
	ICMB	Specifies that the input console message buffer, used to buffer input commands, is to be mapped.
	FCTQ	Specifies that the input console message buffer, used to buffer input commands, is to be mapped.
	OCMB	Specifies that the output console message buffer, used to buffer output messages, is to be mapped.
	OUTPUT	Specifies that the output console message buffer, used to buffer output messages, is to be mapped.
&CODES=		Specifies the ability to set symbolic equates for the various BDT command verbs.

Parameter	Subparameter	Explanation
	YES	Specifies that a set of symbolic equates for the various BDT command verbs are to be generated. This is the default.
	NO	Specifies that the command verb equates are not to be generated.
&PREFIX=		Specifies a prefix to be used in generating labels for the console message area and for the console message codes. The prefix must be a 1- to 3-character string, the first character alphabetic. If three characters are input, certain generated labels will use only the first two characters so that the label generated will not exceed 8 characters. Default prefix values are CONS, CON, and CN.

BDTDDATU

BDTDDATU maps a BDT dynamic allocation text units that are contained in the the variable portion of the MJD.

```
BDTDDATU   &DSECT=  {YES}
                    {NO }
```

Parameter	Subparameter	Explanation
DSECT=	YES	Specifies that the area is to be generated as a DSECT with the label DATUNIT. This is the default.
	NO	Generates space for a text unit in-line, on a byte boundary, with the label DATUNIT.

BDTDGSD

BDTDGSD maps the BDT generalized subtask directory. This macro serves as a control block for the FCT extension (FCT links to the TCB via the GSD) and for generalized ESTAE ABEND service.

```
BDTDGSD       [TYPE=F]
```

Parameter	Subparameter	Explanation
TYPE=F		If TYPE=F is coded, the GSD is built in-line on a full-word boundary. If TYPE=F is not coded, or the parameter is omitted, a DSECT mapping is generated.

BDTDINT

BDTDINT maps the BDT initialization data area. The initialization exit routines (2, 3, 4, 5, and 6) receive control, with register 11 containing the address of the initialization area.

```
BDTDINT
```

BDTDJCT

BDTDJCT maps the job control table entry (JCT) and the scheduler element DSECTs.

```
BDTDJCT
```

BDTDLCT

BDTDLCT maps the BDT logical unit control block (LCT). An LCT is built for each SNA line and BDT node defined to BDT.

```
                 {NODE}
BDTDLCT    &RMT= {NONE}
                 {ALL }
```

Parameter	Subparameter	Explanation
RMT=		Specifies whether the variable portion of the node VLU control block is to be defined.
	NODE	Specifies that the variable portion is to be generated.
	NONE	Specifies that the variable portion is not to be generated.
	ALL	Specifies that the variable portion is to be generated. This is the default.

BDTDMJD

BDTDMJD maps the master job definition control block (MJD). Text units describing the transaction are contained in the MJD.

```
BDTDMJD
```

BDTDREG

BDTDREG maps the symbolic equates for the registers.

```
BDTDREG
```

BDTDRLT

BDTDRLT maps the BDT resident logical unit table (RLT). An RLT is built for each SNA line and BDT node defined to BDT.

```
                 {RESIDENT}
BDTDRLT    &ENTRY= {INISH   }
                 {ALL     }
```

Parameter	Subparameter	Explanation
&ENTRY=		Specifies which sections of BDTDRLT are to be generated.
	RESIDENT	Specifies that only the RLT table for the resident node is to be generated. This is built as a result of processing BDTNODE initialization statements in BDTINR2.
	INISH	Specifies that the initialization RLT table is to be generated. This table is used to process the BDTNODE statements in BDTINR1. **Note:** This specification is only valid for exit routine BDTUX04.
	ALL	Specifies that all sections are to be generated.

BDTDSEQ

BDTDSEQ maps the data CSECT used by BDTSEQ.

```
BDTDSEQ
```

BDTDSMF

BDTDSMF maps the BDT system management facility (SMF) record.

```
BDTDSMF
```

BDTDTVT

BDTDTVT maps the BDT transfer vector table (TVT).

```
BDTDTVT
```

BDTDXOID

BDTDXOID maps a BDT transaction origin ID in its internal format.

```
&label     BDTDXOID &PR=
```

Parameter	Subparameter	Explanation
&label		Optionally specifies a label to be associated with the transaction origin ID.
&PR=		Optionally specifies a 1- to 4-character prefix for each label generated to defined fields within the transaction origin ID.

Chapter 12. Executable Macro Reference

This chapter describes the executable BDT macros that you can use when coding user exit routines. The macros are listed in alphabetic order.

The executable macros identified in this chapter are provided as Product-Sensitive programming interfaces for customers by BDT.

Attention: Do not use as programming interfaces any BDT executable macros other than those identified in this chapter.

Macros can have two types of operands: positional and keyword. A positional operand is written as a string of characters. This character string can be an expression, an implied or explicit address, or some special operand form allowed in a particular macro instruction. Positional operands must be included in a specific order. If a positional operand is omitted and another positional operand is written to the right of it, the comma that would normally have preceded the omitted operand must be included. This comma should be written only if followed by a positional operand; it need not be written if followed by a keyword operand or a blank.

Note: If in the future IBM provides a maintenance PTF that modifies a BDT macro, the entire macro should be replaced at that time with the shipped PTF. Therefore, it is highly recommended that you do not make any modifications to any macros in the macro library. If you do so and want to preserve those modifications following the macro replacement you must reenter those changes.

BDTDKYWD

BDTDKYWD builds a keyword table CSECT. It calls BDTDTUD, a macro which generates BDT text units.

Use the BDTDKYWD macro to define a keyword to be added to the BDT transaction language.

For each keyword you define, the BDTDKYWD macro does two things:

- It generates an entry in a keyword table. This entry contains the keyword and its synonyms, if any. It also contains a pointer to the first or only text unit descriptor for the keyword.
- It calls the BDTDTUD macro, which generates the first or only text unit descriptor for the keyword. (If your keyword is to have more than one associated text unit, you must code a BDTDTUD macro to define the additional text unit descriptors. See the preceding example in the discussion of the LAST parameter.)

  The text unit descriptor contains a skeleton for the text unit to be built, and it describes the syntactic rules for the keyword values and for the text unit parameter values.

The BDT language processor uses the information from the keyword table and the text unit descriptor table to parse the transaction parameters entered by the user and to build the text units that describe the processing that the parameters request.

```
&PRIKEY   BDTDKYWD &SYN=

                          {FIXED}
                  ,&TYPE=  {KEY  }
                          {VAR  }
                          {TERM }
                  ,&TUKEY=
                  ,&TUNUM= 1
                  ,&TULNG= 1
                  ,&TUPAR=
                  ,&TUCPRM=
                  ,&KEYS=
                  ,&MAXLEN= 255
                  ,&PREFIX=

                  ,&OPT=  {YES}
                          {NO }

                  ,&MAXVAL=  x'FFFFFFFF'

                  ,&NUMERIC=  {YES}
                              {NO }

                  ,&CONVERT=  {YES}
                              {NO }

                  ,&LAST=  {YES}
                           {NO }

                  ,&DATU=  {YES}
                           {NO }
```

Parameter	Subparameter	Explanation
&PRIKEY		A required label that specifies the keyword defined by this macro call. This is the statement label coded on the BDTDKYWD macro. Example: `SECPSWD BDTDKYWD...`
&SYN=		Specifies the synonym(s) that may be used instead of the keyword when the keyword is entered on a BDT transaction. The synonyms may be a list.
&TYPE=		A required parameter that specifies the type of keyword to be generated.
	FIXED	Indicates that the keyword is a stand-alone keyword and may have no values. OLD, NEW, and MOD are fixed keywords.

	KEY	Indicates that the keyword has a list of valid values. Only the specified values as defined in the KEYS= operand may be entered with this keyword on a BDT transaction. For example:
		
		`DISP BDTDKYWD TYPE=KEY,`
		`KEYS=(CATLG,UNCATLG)`
		
		When the previous example is coded, the DISP parameter on BDT transactions can only be coded DISP(CATLG) or DISP(UNCATLG).

	VAR	Indicates that the keyword being defined has a value associated with it, and it is variable. For example:
		
		`SECUSER BDTDKYWD TYPE=VAR,`
		`MAXLEN=8,...`
		
		When the previous example is coded, transactions that use the SECUSER parameter, must have a value of up to 8 characters entered with them.

Parameter	Subparameter	Explanation
	TERM	Specifies that the end-of-list terminator be generated. This is a required parameter that is used for the last occurrence of the BDTDKYWD or BDTDTUD macro coded.

&TUKEY		Defines the text unit key for the text unit being defined when TYPE=FIXED, KEY, or VAR. If the text unit is a dynamic allocation text unit, the value specified for this keyword should be one of the dynamic allocation text unit values defined in the IEFZB4D2 macro. If this is a BDT text unit, the value specified should be in the range of 200-220 for nongeneric keywords (those that can be specified in either or both the FROM and TO sections of a transaction). For generic keywords (those that are specified in the job definition section of a transaction and define processing options or the transaction as a whole) select a key value in the range of 491-511.
		
		For example, if the following is coded, the MEMBER keyword has a dynamic allocation text unit with a text unit key equal to the value defined in the IEFZB4D4 macro for DALMEMBR:
		
		`MEMBER BDTDKYWD TYPE=VAR,`
		`  TUKEY=DALMEMBR,DATU=YES,...`

&TUNUM=		Specifies the number of parameter fields to be included in the text unit generated for the keyword. The default is 1. Therefore, if the text unit is to have no parameter fields, TUNUM=0 must be specified. TUNUM generates the count field in the text unit.
&TULNG=		Defines the length of the text unit parameter field, if one is to be included in the text unit generated for this keyword. The default is 1. If there is to be no text unit parameter field, TUNUM=0 must be coded, and TULNG need not be coded. (The dynamic allocation generated by BDT for the ROUND keyword, DALROUND, is an example of a text unit with no parameter.) If the keyword value associated with this text unit is VAR and its value is character (NUMERIC=NO), or if its value is numeric but not to be converted to binary, (NUMERIC=YES,CONVERT=NO), then the actual length of the keyword value entered by the user will be used for the text unit parameter length when the language processor is parsing the transaction parameters and building the text units. For these cases, TULNG=0 should be specified.

Parameter	Subparameter	Explanation
&TUPAR=		Defines the parameter value to be included in this text unit generated for this keyword. TUPAR may be used, for example, when you are defining a TYPE=FIXED keyword whose associated text unit has one defined value. The length of the generated text unit field is determined by the value of the TULNG parameter. TUPAR can be specified as a symbol that has been equated to the desired value. For example, if you have coded USRTUPRM EQU X‘80’, you could then specify TUPAR=USRTUPRM. The resulting text unit parameter field would contain the value X‘80’. The length of the field would be that specified via TULNG. If TULNG was not specified, the length would default to 1.
&TUCPRM=		Defines the parameter value to be included in the text unit generated for this keyword when this value is a character string. TUCPRM may be used, for example, when you are defining a TYPE=FIXED keyword whose associated text unit has one defined character value. For example, the text unit parameter might be the name of a module, MSGROUTE, which another user is to invoke when the user keyword is specified on a transaction. In this case, you could code TUCPRM=MSGROUTE. Note that the character string is not enclosed in single quotes.

&KEYS=		Used with TYPE=KEY to define the valid parameters associated with the keyword. For example, if the following is coded, the only parameters that are valid with the keyword LABEL on a BDT transaction are NL, SL, SUL, or BLP:
		
		`LABEL  BDTDKYWD TYPE=KEY,`
		`       TUKEY=DALLABEL,`
		`       KEYS=(NL,SL,SUL,BLP),...`

&MAXLEN=		Specifies the maximum length of the keyword value for a TYPE=VAR value. MAXLEN=255 is the default. For example, if you were defining a keyword USR(userid) where the userid could be a maximum of 8 characters, you would code MAXLEN=8.

Parameter	Subparameter	Explanation

&PREFIX=		Defines a prefix to be used for the symbols generated to define the text unit parameter values to be associated with the keys specified by the KEYS= parameter for a keyword defined as TYPE=KEY. The prefix may be from one to seven valid assembler language characters. The symbols are generated by concatenating the specific prefix with as many characters of each KEY value as can be used to create a symbol of up to eight characters.
		
		For example, assume that you are defining a user keyword MSGSUPR whose values may be YES, Y, NO, or N:
		
		`MSGSUPR(YES|Y|NO|N)`
		
		The text unit parameter value for YES or Y is to be X‘0200’; the value for NO or N is to be X‘0400’. Your installation naming convention for user text unit parameter values is to begin each name with the characters USR. You could code the following to define the text unit parameter values:
		
		`USRSUPPY EQU X'0200'`
		`USRSUPPN EQU X'0400'`
		
		You could then code the following BDTDKYWD macro:
		
		`MSGSUPR BDTDKYWD TYPE=KEY,`
		`KEYS=(YES,Y,NO,N),`
		`TULNG=2,PREFIX=USRSUPP,...`
		
		TULNG=2 is necessary to define the 2-byte text unit parameter field, since 1 is the default. The previous example would generate the following to define the key values and their associated text unit parameter values:
		
		`DC CL3'YES',AL2(USRSUPPY)`
		`DC CL3'Y',AL2(USRSUPPY)`
		`DC CL3'NO',AL2(USRSUPPN)`
		`DC CL3'N',AL2(USRSUPPN)`

Parameter	Subparameter	Explanation

&OPT=		Specifies whether this keyword value is optional. The default is OPT=NO. If OPT=NO is coded or the parameter is omitted, the value must be specified when the keyword is entered on a BDT transaction.
		
		For example, suppose you want to define a user keyword that specifies whether transaction messages are to be suppressed and the name of a log where messages are to be written by a user exit routine. If the log name is not specified, a default will be used. Syntactically, this keyword might look like the following:
		
		`SMSG(YES|Y|NO|N[,log-name])`
		
		The definition of the keyword could be as follows:
		
		`SMSG BDTDKYWD TYPE=KEY,`
		`KEYS=(YES,Y,NO,NO),`
		`LAST=NO,...`
		`BDTDTUD TYPE=VAR,MAXLEN=8,`
		`OPT=YES,LAST=YES,...`
		
		This definition will cause the value for the log name to be optional.

&MAXVAL=		Specifies the maximum value that a keyword value may have, when it is TYPE=VAR and the keyword value is defined as NUMERIC=YES, CONVERT=YES. If MAXVAL is not specified for a NUMERIC keyword value that is to be converted to binary, the converted value must be less than or equal to X'FFFFFFFF'. Note that the TULNG parameter must specify a text unit parameter length great enough to hold the maximum value.
&NUMERIC=		For a TYPE=VAR keyword value, specifies whether the keyword value must be numeric. The default is NUMERIC=NO. If this parameter is omitted or if NUMERIC=NO is coded, the parameter value can be any valid alpha-national character.
&CONVERT=		For a TYPE=VAR, NUMERIC=YES keyword value, specifies whether the keyword value is to be converted into binary before being placed in the text unit parameter field. The default is CONVERT=NO.

Parameter	Subparameter	Explanation

&LAST=		Specifies whether this is the last text unit to be associated with this keyword. LAST=YES is the default. Use LAST=NO if you want to define more than one text unit to be associated with a particular keyword. Define each additional text unit with the BDTDTUD macro, immediately following the BDTDKYWD macro. Specify (or default) LAST=YES on the last BDTDTUD macro.
		
		For example, if you want to define a keyword for which three text units are to be generated, you would specify the following:
		
		`BDTDKYWD...,LAST=NO`
		`BDTDTUD...,LAST=NO`
		`BDTDTUD...,LAST=YES`
		
		The BDTDKYWD macro defines the keyword and the first text unit. The two BDTDTUD macros define the second and third text units for the keyword.
		
		For example, BDT generates two dynamic allocation text units for the DISP keyword and for the SPACE keyword.

&DATU=		Identifies the text unit as either a dynamic allocation text unit (DATU=YES) or a BDT text unit (DATU=NO). DATU=NO is the default.

BDTDTUD

BDTDTUD creates text units to describe the keywords used in a BDT transaction. This macro is called by BDTDKYWD.

Use the BDTDTUD macro to define additional text unit descriptors when the transaction keyword you are defining has more than one text unit associated with it.

The text unit descriptor contains a skeleton for the text unit which the BDT language processor will build when the associated keyword is entered on a transaction. It describes the syntactic rules for the keyword values and for the text unit parameter values.

```
&label   BDTDTUD            {FIXED}
                  &TYPE=    {KEY  }
                            {VAR  }
                            {TERM }

                  ,&TUKEY=
                  ,&TUNUM= 1
                  ,&TULNG= 1
                  ,&TUPAR=
                  ,&KEYS=
                  ,&MAXLEN= 255
                  ,&PREFIX=

                  ,&OPT=    {YES}
                            {NO }

                  ,&MAXVAL= x'FFFFFFFF'

                  ,&NUMERIC=  {YES}
                              {NO }

                  ,&CONVERT=  {YES}
                              {NO }

                  ,&LAST=   {YES}
                            {NO }

                  ,&DATU=   {YES}
                            {NO }
```

Parameter	Subparameter	Explanation
&label		Optional positional parameter that specifies the label associated with the first statement generated on the macro execution.
&TYPE=		A required parameter that specifies the type of keyword to be generated.
	FIXED	Indicates that the keyword is a stand-alone keyword and may have no values. OLD, NEW, and MOD are fixed keywords.
	KEY	Indicates that the keyword has a list of valid values. Only the specified values as defined in the KEYS= operand may be entered with this keyword on a BDT transaction.
	VAR	Indicates that the keyword being defined has a value associated with it, and it is variable.
	TERM	Specifies that the end-of-list terminator be generated. This is a required parameter that is used for the last occurrence of the BDTDKYWD or BDTDTUD macro coded.
&TUKEY=		Defines the text unit key for the text unit being defined when TYPE=FIXED, KEY, or VAR. If the text unit is a dynamic allocation text unit, the value specified for this keyword should be one of the dynamic allocation text unit values defined in the IEFZB4D2 macro. If this is a BDT text unit, the value specified should be in the range of 200-220 for nongeneric keywords (those that can be specified in either or both the FROM and TO sections of a transaction). For generic keywords (those that are specified in the job definition section of a transaction and define processing options or the transaction as a whole) select a key value in the range of 491-511.

Parameter	Subparameter	Explanation
&TUNUM=		Specifies the number of parameter fields to be included in the text unit generated for the keyword. The default is 1. Therefore, if the text unit is to have no parameter fields, TUNUM=0 must be specified. TUNUM generates the count field in the text unit.
&TULNG=		Defines the length of the text unit parameter field, if one is to be included in the text unit generated for this keyword. The default is 1. If there is to be no text unit parameter field, TUNUM=0 must be coded, and TULNG need not be coded. (The dynamic allocation generated by BDT for the ROUND keyword, DALROUND, is an example of a text unit with no parameter.) If the keyword value associated with this text unit is VAR and its value is character (NUMERIC=NO), or if its value is numeric but not to be converted to binary, (NUMERIC=YES,CONVERT=NO), then the actual length of the keyword value entered by the user will be used for the text unit parameter length when the language processor is parsing the transaction parameters and building the text units. For these cases, TULNG=0 should be specified.
&TUPAR=		Defines the parameter value to be included in this text unit generated for this keyword. TUPAR may be used, for example, when you are defining a TYPE=FIXED keyword whose associated text unit has one defined value. The length of the generated text unit field is determined by the value of the TULNG parameter. TUPAR can be specified as a symbol that has been equated to the desired value. For example, if you have coded USRTUPRM EQU X‘80’, you could then specify TUPAR=USRTUPRM. The resulting text unit parameter field would contain the value X‘80’. The length of the field would be that specified via TULNG. If TULNG was not specified, the length would default to 1.
&KEYS=		Used with TYPE=KEY to specify the parameter keyword value.
&MAXLEN=		Specifies the maximum length of the keyword value for a TYPE=VAR value. MAXLEN=255 is the default. For example, if you were defining a keyword USR(userid) where the userid could be a maximum of 8 characters, you would code MAXLEN=8.
&PREFIX=		Defines a prefix to be used for the symbols generated to define the text unit parameter values to be associated with the keys specified by the KEYS= parameter for a keyword defined as TYPE=KEY. The prefix may be from one to seven valid assembler language characters. The symbols are generated by concatenating the specific prefix with as many characters of each KEY value as can be used to create a symbol of up to eight characters.
&OPT=		Specifies whether this keyword value is optional. The default is OPT=NO. If OPT=NO is coded or the parameter is omitted, the value must be specified when the keyword is entered on a BDT transaction.

Parameter	Subparameter	Explanation
&MAXVAL=		Specifies the maximum value that a keyword value may have, when it is TYPE=VAR and the keyword value is defined as NUMERIC=YES, CONVERT=YES. If MAXVAL is not specified for a NUMERIC keyword value that is to be converted to binary, the converted value must be less than or equal to X'FFFFFFFF'. Note that the TULNG parameter must specify a text unit parameter length great enough to hold the maximum value.
&NUMERIC=		For a TYPE=VAR keyword value, specifies whether the keyword value must be numeric. The default is NUMERIC=NO. If this parameter is omitted or if NUMERIC=NO is coded, the parameter value can be any valid alpha-national character.
&CONVERT		For a TYPE=VAR, NUMERIC=YES keyword value, specifies whether the keyword value is to be converted into binary before being placed in the text unit parameter field. The default is CONVERT=NO.
&LAST=		Specifies whether this is the last text unit to be associated with this keyword. LAST=YES is the default. Use LAST=NO if you want to define more than one text unit to be associated with a particular keyword. Define each additional text unit with the BDTDTUD macro, immediately following the BDTDKYWD macro. Specify (or default) LAST=YES on the last BDTDTUD macro.
&DATU=		Identifies the text unit as a dynamic allocation text unit (DATU=YES) or a BDT text unit (DATU=NO). DATU=NO is the default.

BDTXASRV

BDTXASRV invokes abend services during abend recovery processing. This service must be invoked as part of abend recovery processing so that a task can continue to run. Abend services cannot clean up after a task. It is the responsibility of the abending task to properly release all acquired resources. Failure to do so can seriously impact the proper functioning of BDT. Abend services are only available to functions running in the BDT address space.

```
&label  BDTXASRV      &TYPE=  {EXIT }
                              {RETRY}
                      ,&GSD=
                      ,&SDWA=
                      ,&SAPURGE= (R13)
                      ,&NORMAL=
                      ,&RESTART= *
                      ,&TERM= *,
                      ,&RCVMOD= BDTABMN
                      ,&RCVLABEL= BDTABMN
```

Parameter	Subparameter	Explanation
&label		Optional label associated with the first instruction generated by the BDTXASRV macro.

Parameter	Subparameter	Explanation
&TYPE=		Specifies whether abend services is being invoked for ESTAE exit routine processing (TYPE=EXIT), or for ESTAE retry routine services (TYPE=RETRY). For proper recovery, the ESTAE exit routine should issue this macro with TYPE=EXIT, which records the abend environment and issues an SVC dump if necessary, then issue this macro a second time in the retry routine, which performs general cleanup and provides a BDT formatted dump, if necessary.
&GSD=		Specifies the address of the generalized subtask directory (GSD) for the task for which abend services is invoked.
&SDWA=		Specifies the address of the system diagnostic work area (SDWA).
&SAPURGE=		Specifies the address of the lowest save area to be retained for the function following abend services processing. Abend services frees all save areas preceding the specified save area. This ensures that BDT save areas trapped by an abend will not be permanently lost. The default address is in register 13.
&NORMAL=		Specifies the exit to be taken if normal abend recovery is permitted. The normal exit address may be contained in a register. If this exit receives control, the function may recover from the abend and resume processing. The function must ensure that the environment is left with all data sets closed, all system acquired resources returned to the system.
&RESTART=		Specifies the exit to be taken if the operator requests that the executing function be canceled and subsequently restarted. The restart address may be contained in a register. If the exit receives control, the function must perform any required clean-up. The function must then return to the address 4 bytes past the address contained in TVT JSSRT. This applies to functions running under control of DAPs only (i.e. jobs), not resident functions. Resident functions should specify the same address for RESTART and TERM, and proceed as for TERM. This parameter is required if TYPE=RESTART is coded.
&TERM=		Specifies the exit be taken if recovery from the abend is not possible. The termination exit address may be contained in a register. If this exit receives control, the function must perform any required clean-up operations and exit. This parameter is required if TYPE=RESTART is coded.
&RVCMOD=		Specifies the name of the recovery routine (ESTAE) that issues the BDTXASRV macro. The default is BDTABMN.
&RCVLABEL=		Specifies the name of the recovery routine (ESTAE exit) label that issues the BDTXASRV macro. The default is BDTABMN.

BDTXJCT

BDTXJCT provides for the controlled access to the job's job control table (JCT). The macro can be used to add a new job and its JCT, modify the JCT, read the JCT, delete a JCT from the system queue, or purge a job and its JCT from the system.

```
&label    BDTXJCT  &TYPE=
                   ,&JOB=
                   ,&ERROR=
                   ,&BUSY=
                   ,&NORMAL=
                   ,&DEQ=
```

Parameter	Subparameter	Explanation
&label		An optional name to be associated with the first instruction generated by this macro.
&TYPE=		Specifies the type of access requested by this macro call. The JCT may be any one of the following:
	ADD	Add a new job and its corresponding JCT to the system queue.
	RW	Obtain read/write access to a JCT to modify or update it.
	RO	Obtain read-only access to a JCT.
	DEL	Delete a JCT from the system queue.
	REL	Relinquish read-only or read/write access to a JCT previously obtained using the TYPE=RO or TYPE=RW specifications, respectively.
	PURGE	Purge a job from the system.
&JOB=		Specifies the JCT to be added if TYPE=ADD is specified or accessed. For all other TYPE= specifications the JCT may be any of the following: 1. The job number. 2. The address of an area containing the job number (1-4 EBCDIC digits terminated by one or more blanks). 3. The address of an area containing the job name (1-8 characters terminated by one or more blanks). 4. The address of the job queue element (JQE). If register notation is used, the specified register must contain an appropriate address or job number. If register 1 is specified, register loading is suppressed.
&ERROR=		Specifies the return point that receives control if an error is detected while performing the required function. If register notation is used, the register must contain the address of the appropriate exit routine.
&BUSY=		Specifies the address of a return point if the access to a requested priority level is not immediately available. If register notation is used, the register must contain the address of the appropriate exit routine. The use of this keyword is optional; if it is not specified and access to the specified priority level cannot be granted immediately, the requestor is put into a wait state (by AWAIT) until access can be granted.
&NORMAL=		Specifies the return point that receives control if the called routine returns to the calling routine specifying that normal processing has occurred. A branch to the specified address is automatically taken following linkage to the called routine.

Parameter	Subparameter	Explanation
&DEQ=	ONE ALL	Specifies the priority level (DEQ=ONE) or levels (DEQ=ALL) to be dequeued that were previously enqueued by use of the PRTY= keyword.

BDTXJQE

BDTXJQE provides for the controlled access to the job queue element (JQE). The initial call returns the first JQE within a specified priority level.

```
&label     BDTXJQE      &JOB=
                        ,&PRTY=
                        ,&ERROR=
                        ,&BUSY=
                        ,&NORMAL=
                        ,&DEQ=
```

Parameter	Subparameter	Explanation
&label		An optional name to be associated with the first instruction generated by this macro.
&JOB=		Specifies the JQE to be accessed. The JQE may be any of the following: 1. The job number (binary). 2. The address of an area containing the job number (1-4 EBCDIC digits terminated by one or more blanks). 3. The address of an area containing the job name (1-8 characters terminated by one or more blanks or an asterisk). The asterisk must to coded in columns 2 through 7; it specifies that this is a generic search for any job name containing the specified character string. This parameter is mutually exclusive with the use of the &PRTY= keyword.
&PRTY=		Specifies the particular priority level to be searched by this macro call. The initial call returns the address of the JQE for the first job within the specified priority level. Each subsequent call returns the address of the JQE for the next job within the specified priority level.
&ERROR=		Specifies the return point that receives control if an error is detected while performing the required function. If register notation is used, the register must contain the address of the appropriate exit routine.
&BUSY=		Specifies the address of a return point if the access to a requested priority level is not immediately available. If register notation is used, the register must contain the address of the appropriate exit routine. The use of this keyword is optional; if it is not specified and access to the specified priority level cannot be granted immediately, the requestor is put in a wait state (by AWAIT) until access can be granted.
&NORMAL		Specifies the return point that receives control if the called routine returns to the calling routine specifying that normal processing has occurred. A branch to the specified address is automatically taken following linkage to the called routine.

Parameter	Subparameter	Explanation
&DEQ=	ONE ALL	Specifies the priority level (DEQ=ONE) or levels (DEQ=ALL) to be dequeued that were previously enqueued by use of the PRTY= keyword.

BDTXTRC

BDTXTRC adds events to the trace table.

```
&label    BDTXTRC  ,ID=id

                   ,REG=n

                              {{(Rn)       }}
                              {{field      }}
                   ,TWm   =   {{offset(Rn)}}    ...

                              {{=C'data'   }}
```

Parameter	Subparameter	Explanation
&label		An optional label associated with the first instruction generated by the BDTXTRC macro.
ID=*id*		An optional parameter that is a one-character identification for this trace event. This ID appears in the trace table entry to identify which BDTXTRC macro expansion created the entry.
REG=*n*		An optional parameter that specifies the register that will contain the address of the GSD. If the BDTXTRC macro fails, you can restore the registers that are saved in BSDERREG. *n* ranges from 1 to 15.
TW*m*=		An optional parameter that is a fullword field of data that resides in the trace table entry. *m* is an integer from 1 to 6. Code *TWm* in ascending order: TW1, TW2, TW3, and so on.
	(R*n*)	Indicates the register that substitutes directly as the first operand of an ST instruction. *n* is a number from 1 to 15.
	field	Indicates the name of a field. This parameter substitutes directly as the second operand of an MVC instruction.
	offset(R*n*)	Indicates an offset register. This parameter substitutes directly as the second operand of an MVC instruction.
	=C'*data*'	Indicates the actual data. Use two equal signs between TW*m* and C, and enclose the data in quotation marks. This parameter substitutes directly as the second operand of an MVC instruction.

Include the IHAPSA, IKJTCB, BDTDGSD, and BDTDTWA mapping macros when you use the BDTXTRC macro.

If you do not code any parameters in the BDTXTRC macro, the associated fields in the trace entry contain zeroes.

The fields in the trace table entries are built by BDTGRTX from data in the trace work area (TWA). These fields reflect the coding of the BDTXTRC macro. See Table 8 on page 157.

Table 8. BDT Trace Table

Offset	Length	Name	Description
0	4 bytes	TRCNAME	Bytes 4-7 of name of module that issued the BDTXTRC macro
4	1 byte	TRCID	Unique trace event ID
5	1 byte		Reserved
6	2 bytes	TRCOFF	Offset of BDTXTRC macro in issuing module
8	4 bytes	TRCTW1	TW1 unique fullword field
C	4 bytes	TRCTW2	TW2 unique fullword field
10	4 bytes	TRCTW3	TW3 unique fullword field
14	4 bytes	TRCTW4	TW4 unique fullword field
18	4 bytes	TRCTW5	TW5 unique fullword field
1C	4 bytes	TRCTW6	TW6 unique fullword field
20	4 bytes	TRCR00	Register 0 of task when BDTXTRC was issued
24	4 bytes	TRCR01	Register 1 of task when BDTXTRC was issued
28	4 bytes	TRCR10	Register 10 of task when BDTXTRC was issued
2C	4 bytes	TRCR11	Register 11 of task when BDTXTRC was issued
30	4 bytes	TRCR14	Register 14 of task when BDTXTRC was issued
34	4 bytes	TRCR15	Register 15 of task when BDTXTRC was issued
38	4 bytes	TRCTCBT	Address of the TCB under which the trace entry was created
3C	4 bytes	TRCTIM	Time of day when the trace entry is made

As an example of specifying the BDTXTRC macro, a trace entry with the following data:

- An ID of 10
- Register 4 as a work register
- TW1 containing the character string ‘SSI’,
- TW2 containing the contents of a field called CPID
- TW3 containing the contents of register 8
- TW4 containing the contents of the fullword pointed to by register 8 when you issued the trace BDTXTRC macro

could be created with this BDTXTRC macro:

```
BDTXTRC ID=10,REG=4,TW1==C'SSI',TW2=CPID,TW3=(R8),TW4=0(R8)
```

BDTXTUAM

BDTXTUAM retrieves text units in an MJD control block. The address of the located text unit is returned in register 1.

Note: BDTXTUAM can be used to access text units from exit routine BDTUX19 only when BDTUX19 runs in the BDT address space. Prior to using BDTXTUAM you must establish addressability to the TVT.

```
&label  BDTXTUAM      &KEY

                      ,&TYPE=  {FROM}
                               {TO  }

                      ,&START=  {label     }
                                {(register)}

                      ,&EOD=  {label     }
                              {(register)}

                      ,&NORMAL=  {label     }
                                 {(register)}

                      ,&DATU=  {YES}
                               {NO }

                      ,&WORKA=
```

Parameter	Subparameter	Explanation
&label		Optional label associated with the first instruction generated by the BDTXTUAM macro.
&KEY=		Specifies the value of the key for the text unit to be returned. If no key is specified (or is zero) the next text unit of the specified type is returned.
&TYPE		Specifies the type of text unit to be returned.
		Note: If this keyword is not specified, the returned text unit could be either source or destination text units; this depends upon which is found first.
	FROM	Indicates the text units to be returned are source text units.
		Indicates the text units to be returned are destination text units.
&START=		Specifies the address where the search is to begin. If the address is the address of an MJD, the search begins with the first text unit in the MJD. If the address is a text unit, the search begins with the succeeding text unit. The START address may be contained in a register or an A-type address. This parameter is required.
&EOD=		Specifies the return point given control if the called routine returns to the calling routine specifying an end of data condition.
&NORMAL=		Specifies the return point given control if the called routine returns to the calling routine specifying the occurrence of normal processing.
&DATU=		Specifies the type of text unit to be returned as either a dynamic allocation text unit or a BDT text unit, or both. The default is either.
	YES	Specifies a dynamic allocation text unit.
	NO	Specifies a BDT text unit.
		Note: If this keyword is not specified, the returned text unit could be either dynamic allocation or BDT text units; this depends upon which is found first.

Parameter	Subparameter	Explanation
&WORKA=		Specifies the address of a work area to be used to build a parameter list. **Note:** If this keyword is not specified, a parameter list will be generated inline that cannot be used in reentrant modules such as user exits. It becomes the user's responsibility to provide a unique work area.

Appendix A. Parameter map

This appendix summarizes the matchups that must exist between BDT, MVS, and VTAM parameters.

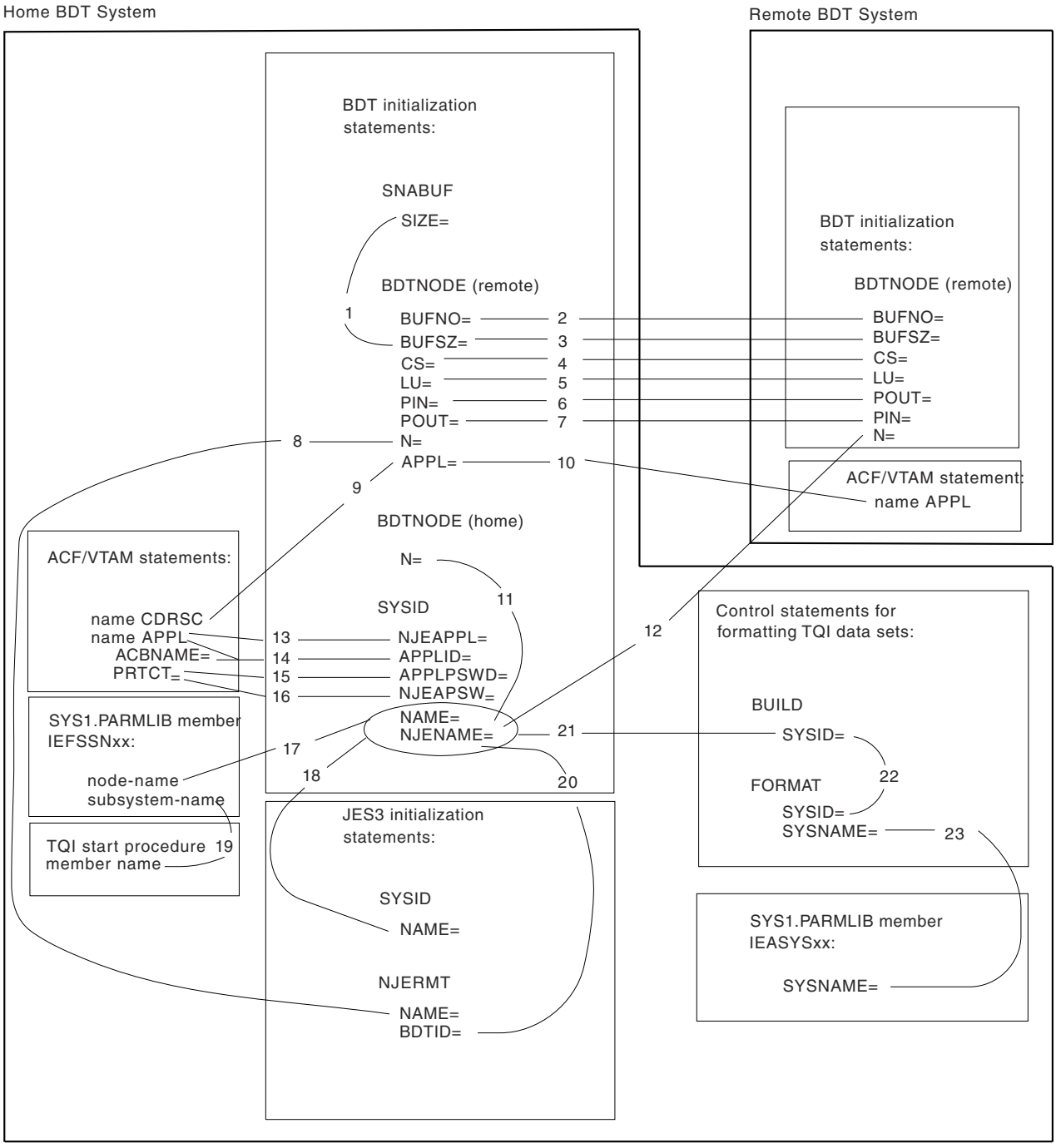


Figure 31. Matchups between BDT, MVS, and VTAM parameters

The following explanations apply to Figure 31 on page 161:

1. SIZE must be equal to or greater than BUFSZ.
2. If different values are specified for BUFNO, BDT uses the lesser of the two.
3. If different values are specified for BUFSZ, BDT uses the lesser of the two.

4. The CS values must match in order for data compression to occur. See Table 3 on page 41.
5. If different values are specified for LU, BDT uses the lesser of the two.
6. The passwords specified on PIN and POUT must match.
7. The passwords specified on POUT and PIN must match.
8. If the home subsystem has a SNA NJE node, the name on the NAME parameter of the JES3 NJERMT statement must match the name on the N parameter of the BDT BDTNODE statement.
9. The name (label) on CDRSC must match the name specified by APPL.
10. The name specified on the APPL parameter of BDTNODE must match the name (label) on the VTAM APPL statement.
11. The file-to-file node names specified on N and NAME must match.
12. The SNA NJE node names specified on NJENAME and N must match.
13. The name (label) on APPL and the name specified on NJEAPPL must match. Note that NJEAPPL is used only for SNA NJE nodes.
14. If ACBNAME is coded, the name specified for APPLID must match it and a name (label) on APPL is not required. If ACBNAME is omitted, the name specified for APPLID must match the name (label) on APPL. Note that APPLID is used only for file-to-file nodes.
15. The passwords specified on PRTCT and APPLPSWD must match. Note that APPLPSWD is used only for file-to-file nodes.
16. The passwords specified on PRTCT and NJEAPSWD must match. Note that NJEAPSWD is used only for SNA NJE nodes.
17. If IEFSSN*xx* defines a subsystem with a file-to-file node, without or in addition to a SNA NJE node, the node name on IEFSSN*xx* must match the SYSID NAME parameter. If IEFSSN*xx* defines a subsystem with only a SNA NJE node, the node name on IEFSSN*xx* must match the SYSID NJENAME parameter.
18. If the home subsystem has a file-to-file node, without or in addition to a SNA NJE node, the name on the NAME parameter of the JES3 SYSID statement must match the name on the NAME parameter of the BDT SYSID statement. If the home subsystem has a SNA NJE node only, the name on the NAME parameter of the JES3 SYSID statement must match the name on the NJENAME parameter of the BDT SYSID statement.
19. The subsystem name on IEFSSN*xx* must be the last one to four characters of the TQI start procedure member name.
20. If BDT has the SNA NJE feature, BDTID must match NJENAME.
21. If the home subsystem has a file-to-file node, without or in addition to a SNA NJE node, the node name on the SYSID parameter of the BUILD statement must match the SYSID NAME parameter. If the home subsystem has a SNA NJE node only, the node name on SYSID BUILD must match the SYSID NJENAME parameter.
22. The SYSID names on the BUILD and FORMAT control statements must match. (BUILD formats the checkpoint and bit-map data sets, and FORMAT formats message data sets.)
23. The name on the SYSNAME parameter in IEASYS*xx* must match the name on the SYSNAME parameter of the FORMAT control statement.

Note: BDT does not require the SNALINE statement; BDTNODE can be used instead. But if you do use SNALINE (which can be used for file-to-file nodes only) you should know that the following relationships, which are not shown in the figure, apply:

- The node names specified on SNALINE,NODE= at your node and BDTNODE,N= at your node must match.
- The name specified on SNALINE,N= at the home node and the name (label) specified on the APPL statement at the remote node must match.

Appendix B. Virtual Storage Required for the BDT Address Space

The virtual storage required for the BDT address space is the sum of the virtual storage required for modules plus the virtual storage required for data areas, as shown in the following table. (All values are in decimal.)

Activity	Virtual Storage Required for Modules	Virtual Storage Required for Data Areas
Starting BDT	277K bytes[See note 1]	19K bytes + (300 bytes × no. of BDTNODE statements) + (136 bytes × total no. of VLUs) – (216 bytes if file-to-file)[See note 2]
Activating the SNA manager	58K bytes	1K bytes + (644 bytes × (no. of file-to-file BDTNODE statements – 1)) + (644 bytes × no. of SNA NJE BDTNODE statements) + (112 bytes × total no. of VLUs) + (32 bytes × total no. of VLUs)[See note 3]
Transferring a sequential data set	7K bytes[See note 4]	(2419 bytes × no. of active transfers) + (376 bytes × no. of active transfers) + (408 bytes × no. of active transfers)
Transferring a partitioned data set	10K bytes[See note 4]	(990 bytes × no. of active transfers) + (376 bytes × no. of active transfers) + (408 bytes × no. of active transfers)
Sending a SNA NJE job or output	8K bytes[See note 4]	(975 bytes × no. of active transfers) + (376 bytes × no. of active transfers) + (408 bytes × no. of active transfers)
Receiving a SNA NJE job or output	6K bytes[See note 4]	(1171 bytes × no. of active transfers) + (376 bytes × no. of active transfers) + (408 bytes × no. of active transfers)

Notes:

1. 277K bytes includes:
   - 25K bytes for the following modules in the PLPA: BDTLP, BDTSSBDT, BDTSSEOM, BDTSS34, and IGX00034.
   - 5K bytes for the sample required (authorization) exit routines provided by IBM in SYS1.SBDTSAMP: BDTUX25, BDTUX26, BDTUX27, BDTUX28, BDTUX29, and BDTUX31.
2. A more detailed formula is 19K bytes +:
   - For the LCT control block: (136 bytes × (no. of file-to-file BDTNODE statements – 1) + (no. of VLUs)) + (136 bytes × (no. of SNA NJE BDTNODE statements + no. of VLUs))
   - For the RLT control block: (84 bytes × (2 × no. of file-to-file BDTNODE statements – 1)) + (84 bytes × (2 × no. of SNA NJE BDTNODE statements))
3. A more detailed formula is 1K bytes +:
   - For the LCB control block: (644 bytes × (no. of file-to-file BDTNODE statements – 1)) + (644 bytes × no. of SNA NJE BDTNODE statements)
   - For the DCL and LCTE control blocks: (144 bytes × no. of file-to-file VLUs specifying compression) + (112 bytes × no. of file-to-file VLUs without compression) + (144 bytes × no. of SNA NJE VLUs)
4. This storage is for DAP modules, which are kept in storage after they are used the first time. Subsequent transfers do not cause them to be reloaded.

Additional storage is required for the CSA (which VTAM and JES3 use as staging areas and for reading and writing to spool), SQA, LSQA, SWA, and subpools 229-230. Cell pool storage is allocated from user subpools 10-21 (see Table 5 on page 48). This additional storage is configuration and task dependent.

Appendix C. Moving Transactions to a New TQI Checkpoint Data Set

After you start to use BDT you may find that there is insufficient space on the TQI checkpoint data set. If this is the case you may replace the existing checkpoint data set and bit-map data set with larger data sets. To do this you must:

1. Allocate the new data sets.
2. Format the new data sets.
3. Use the BDTTQBCH program to transfer transactions that remain on the existing checkpoint data set to the new checkpoint data set. Include a MOVE control statement in the BDTTQBCH input stream.

Sample JCL and a MOVE control statement are shown in Figure 32 on page 165. Note that the sample assumes that the SBDTLINK library is in the link list and that all data sets are cataloged.

```
//TQIMOVE   EXEC PGM=BDTTQBCH
//OLDBTMP   DD   DISP=SHR,DSN=BDT1.TQIBITS
//OLDDATA   DD   DISP=SHR,DSN=BDT1.TQIDATA
//BITMAPS   DD   DISP=SHR,DSN=BDT1.TQIBITS2
//DATAFILE  DD   DISP=SHR,DSN=BDT1.TQIDATA2
//SYSUDUMP  DD   SYSOUT=A
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
MOVE,SYSID=SYSA1
```

Figure 32. Sample JCL to Move Transactions to a New Checkpoint Data Set

//TQIMOVE EXEC
identifies BDTTQBCH as the program to be executed.

//OLDBTMP DD
defines the old bit-map data set.

//OLDDATA DD
defines the old checkpoint data set.

//BITMAPS DD
defines the new bit-map data set.

//DATAFILE DD
defines the new checkpoint data set.

//SYSUDUMP DD
defines the data set where a formatted storage dump is to be written in the event BDTTQBCH abnormally terminates. You may replace the SYSUDUMP DD statement with a SYSMDUMP DD statement or a SYSABEND DD statement depending on the type of dump you want.

//SYSPRINT DD
defines the data set where BDTTQBCH is to write its messages.

//SYSIN DD
defines the input to BDTTQBCH.

MOVE
is the control statement that moves the transactions to a new checkpoint data set. It can have two parameters:

,SYSID=*node-name*
identifies the BDT node that will read the transactions that TQI writes on the checkpoint data set. *node-name* must match the name on one of the following SYSID initialization statement parameters:

- The NAME parameter if the BDT subsystem has a file-to-file node, without or in addition to a SNA NJE node
- The NJENAME parameter if the BDT subsystem has only a SNA NJE node.

This parameter is required. The example uses SYSA1.

,RECORDS=*nnnn*
defines the number of records that are to be moved from the old checkpoint data set to the new checkpoint data set. If you omit this parameter, BDTTQBCH moves all of the records that are in the old checkpoint data set. The example omits this parameter.

Appendix D. SNALINE Statement (File-to-File Feature Only)

The SNALINE statement can be used to provide VTAM-related information about the connection to a remote file-to-file node.

In BDT the preferred way of providing this information is on the BDTNODE statement, which has been expanded, instead of on SNALINE. Using BDTNODE you can specify all information for a remote node on a single statement. However, SNALINE is still supported so that those who used it in Version 1 initialization streams can use those same initialization streams without change for Version 2 file-to-file initializations.

If you use SNALINE you must code a separate SNALINE statement for each remote node that you have identified on a BDTNODE statement. The maximum number of SNALINE statements in one initialization stream is 100.

SNALINE Statement

```
SNALINE,N=lu-name
    ,NODE=node-name
    [,A={YES|NO}]
    [,ASR={YES|NO|restrt-limit}]
```

N=*lu-name*
identifies the ACF/VTAM logical unit (LU) that the VTAM system programmer at the remote node has defined for BDT. *lu-name* must match the label on the VTAM APPL statement that defines BDT at the remote node. You must specify this parameter; there is no default.

NODE=*node-name*
identifies the remote node to which this statement pertains. *node-name* must be one to eight alphanumeric characters and must match the N parameter on the BDTNODE statement that you coded for this remote node. You must specify this parameter; there is no default.

A={YES|NO}
defines whether BDT is to automatically start a session with the remote node named on this statement. If you select this option, BDT automatically starts a session with the remote node each time the operator at the home node starts BDT and activates the BDT SNA manager.

YES
instructs BDT to automatically start a session.

NO
makes the operator responsible for starting the session. This is the default.

ASR={YES|NO|*restrt-limit* }
defines whether BDT automatically tries to restart a session with a remote node that has failed. BDT can try to restart failing sessions with the remote node with the exception of sessions that failed because they were canceled by the BDT CANCEL command, and sessions that could not start because of a negotiable bind disagreement between the home node and the remote node.

YES
requests that BDT try automatic session restart an unlimited number of times. This is the default.

NO
requests that BDT not attempt automatic session restart.

restrt-limit
requests that BDT try automatic restart a limited number of times. *restrt-limit* defines the number of times that BDT may try to restart a session before giving up. The variable may be a decimal number from 1 to 32767.

Appendix E. Initialization Flow and User Exit Routines

Flow diagram for initialization exits

The following diagram shows how user exit routines fit into the initialization process in BDT.

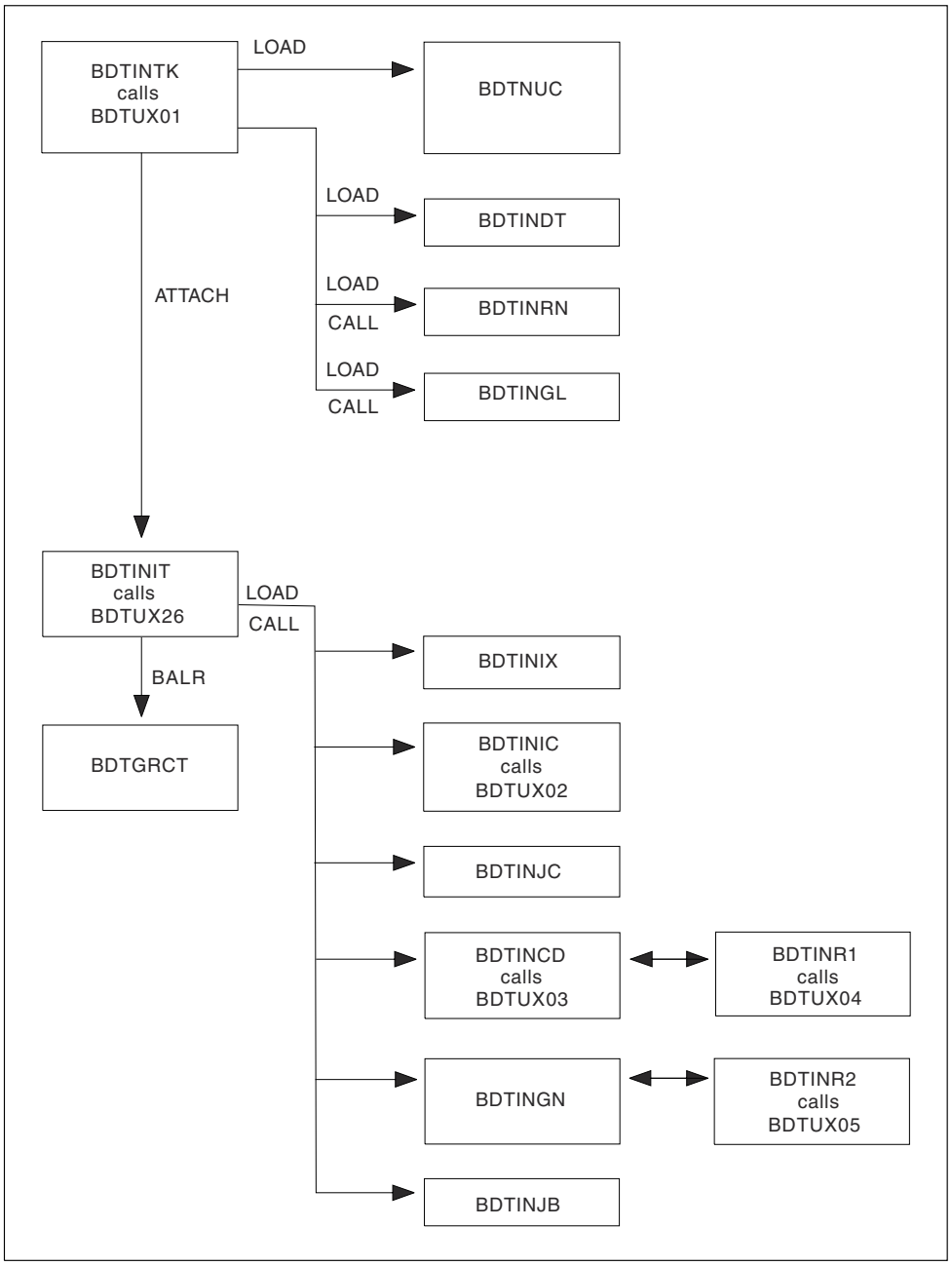


Figure 33. Flow diagram for initialization exits

Internal to External Conversion of the XOID Format

The following steps describe how the XOID format is converted from internal to external format for messages output to the origin of a transaction or command:

1. A BDT module builds a message to send to the transaction origin (XOID). Part of the job of building the message includes the conversion of the XOID from internal to external format. BDTXXOID is called.

External to internal conversion of the XOID format

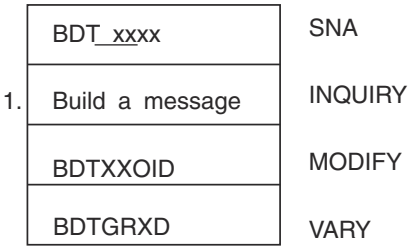
1. The user issues a MODIFY (F) or MESSAGE (Z) command:

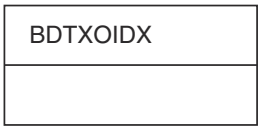
```
BDT F LOG ADEST=(SPK01,SYS1,TSO,CCJMN)
```

   or

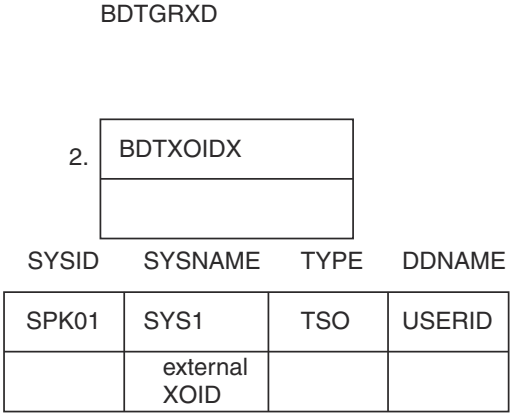
```
BDT Z (SPK02,SYS1,TSO,CN01)Hi,LLOYD!
```

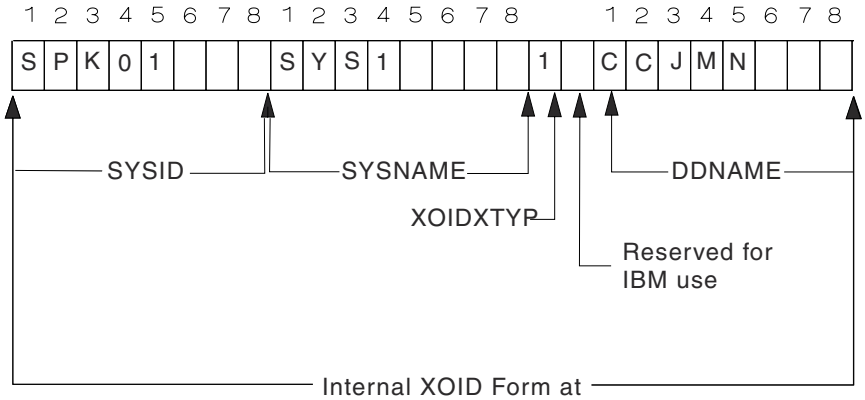
2. The external XOID conversion of SYSID, SYSNAME, TYPE, and DDNAME to internal format is performed by calling BDTXXOID.
3. SYSID and DDNAME are formatted without change. The type field of the external format is converted to its XOIDTYPE code. In this case, TSO is converted to 1.





4. BDTXXOID converts the internal format (SYSID, SYSNAME, TYPE, DDNAME) into the processor name, user type, and identification.





Flow diagram for the invocation of BDTUX17 (job start)

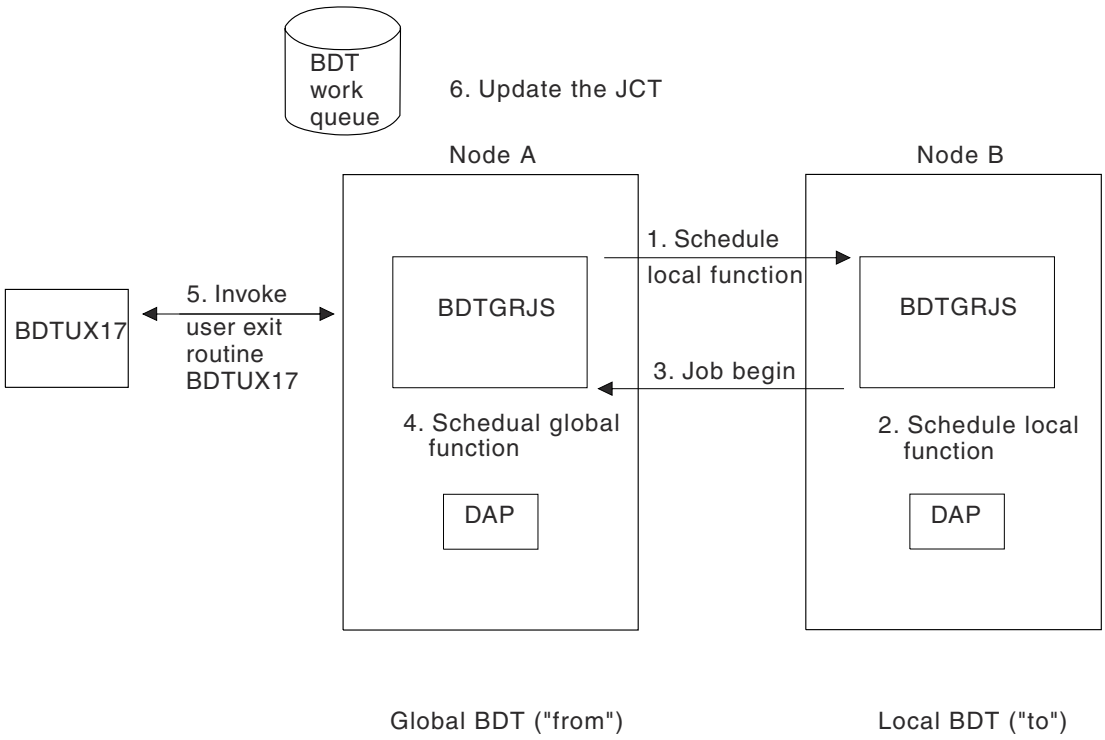


Figure 34. Flow diagram for the invocation of BDTUX17 (job start)

Flow diagram for the invocation of BDTUX18 (job end)

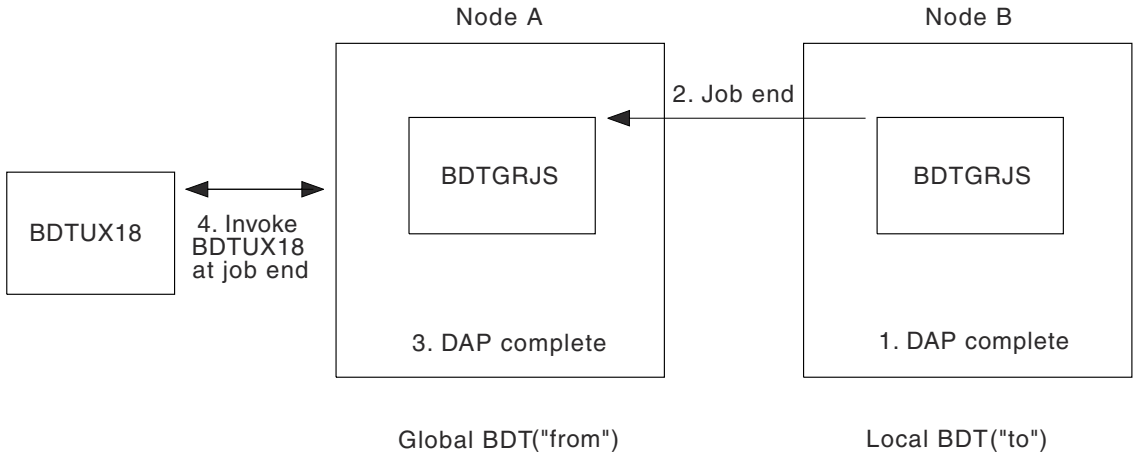


Figure 35. Flow diagram for the invocation of BDTUX18 (job end)

Modules That Issue the BDTXXOID Macro

The following table shows the modules that issue the BDTXXOID macro, and the associated type of conversion.

Issuing Module	Source of Code	Type of Conversion
BDTCMDV	MESSAGE (Z) command Message text	External to internal Internal to external
BDTGRJR	Message text	Internal to external
BDTGRLG	Message text	Internal to external
BDTGRXD	Message text	Internal to external
BDTIQQU	Message text	Internal to external
BDTMDLG	F LOG,ADEST=(SYSID,SYNAME,TYPE,DDNAME) F LOG,ADEST=(SYSID,SYSNAME,TYPE,DDNAME) Message text	External to internal Internal to external Internal to external
BDTSMAP	XOID = parameter	External to internal
BDTSNA	Message text S SNA LOG=(SYSID,SYSNAME,TYPE,DDNAME)	Internal to external External to internal

Appendix F. Sample User Exit Routine

This appendix provides a sample user exit subroutine for BDTUX19. This sample routine locates a specific text unit in the MJD when BDTUX19 is running in the user address space. You can modify this subroutine to meet the needs of your BDT subsystem.

This subroutine requires the following BDT macros and CSECTs:

- BDTDDATU macro (define this macro to map the DATUNIT)
- BDTDMJD macro (MJD macro)
- A user data CSECT similar to the following example:

```
FINDDSTU  DSECT
SAVEAREA  DS   18F     SAVE AREA USED ON ENTRY TO
                       USER EXIT 19 (BDTUX19)
SAVEAREA2 DS   18F     SAVE AREA USED ON ENTRY TO
                       FINDTU
FINDTTYP  DC   X'00'   FIELD TO PASS 'TO' OR 'FROM' PARM
FINDTFRM  EQU  X'80'   'FROM' SIDE TEXT UNIT REQUEST
FINDTTO   EQU  X'40'   'TO' SIDE TEXT UNIT REQUEST
FINDTKEY  DC   F'0'    REQUESTER'S KEY VALUE
```

Note: You must provide a save area. Register 1 is not saved when FINDTU is called.

Assembler Code for Sample Routine

```
CALLING SEQUENCE

MVI       FINDTTYP,FINDTTO             ('TO' TEXT UNIT)
          (OR)
MVI       FINDTTYP,FINDTFRM            ('FROM' TEXT UNIT)
LA        RX,=X'F9'                    SECPSWD KEY TYPE OR OTHER TYPE
                                       KEY VALUES ARE MAPPED IN THE
                                       MACRO EXPANSION OF THE MJD
                                       (BTUXXXX).
ST        RX,FINDTKEY                  WORD CONTAINING KEY TYPE
LA        R15,FINDTU                   ADDRESS OF SUBROUTINE
BALR      R14,R15                      BRANCH TO FINDTU SUBROUTINE

                                       ON RETURN, R1=POINTER TO THE
                                       PASSWORD TEXT UNIT. R1=00000000
                                       IF THE TEXT UNIT IS NOT FOUND.
                                       R15=0 IF THE TEXT UNIT IS FOUND.
                                       R15=4 IF THE TEXT UNIT IS NOT
                                       FOUND.

FINDTU    DS    0H
          STM   R14,R12,SAVAREA2+12    SAVE REGISTERS IN FINDTU'S
                                       SAVE AREA
          L     R4,FINDTKEY            GET KEY TYPE
          LA    R1,MJDSTART            MJD ADDRESS
          AH    R1,MJDFXDLN            BUMP DOWN TO THE TEXT UNITS
          SLR   R2,R2                  CLEAR REGISTER 2
          SLR   R3,R3                  CLEAR REGISTER 3
          B     FINDT400               GO CHECK FIRST KEYWORD
FINDT100  DS    0H
          USING DATUNIT,R1             MAP THE DATUNIT
          CLC   DATUKEY,=X'FFFF'       END OF TEXT UNITS?
          BE    FINDT800               YES -- GO RETURN 'NOT FOUND'
          CLC   DATUNUM,=X'0000'       ANY PARAMETERS?
          BNE   FINDT200               YES, GO GET NUMBER OF PARMS
          LA    R1,DATUENT             NO, GO GET NEXT TEXT UNIT
          B     FINDT400               GO SEE IF THIS IS OUR KEY
FINDT200  DS    0H
          ICM   R2,B'0011',DATUNUM     GET THE NUMBER OF PARMS
          LA    R1,DATUENT             GET ADDRESSABILITY TO PARMS
          USING DATUENT,R1
FINDT300  DS    OH
          ICM   R3,B'0011',DATULNG     GET TEXT UNIT LENGTH
```

```
          LA    R1,DATUPAR(R3)          POINT TO NEXT PARM ENTRY
          BCT   R2,FINDT300             IF MORE PARMS, GO TO FINDT300
FINDT400  DS    OH
          USING DATUNIT,R1
          CLC   DATUKEY,=X'FFFF'        IS THIS THE END OF TEXT?
          BE    FINDT800                YES -- GO RETURN 'NOT FOUND'
          TM    FINDTTYP,TYPETFRM       REQUEST FOR 'FROM' TEXT UNIT?
          BZ    FINDT500                NO, GO TO FINDT500
          TM    DATUFLG,DATUDST         IS THIS A 'FROM' TEXT UNIT?
          BZ    FINDT100                NO, GET NEXT TEXT UNIT
          B     FINDT600                GO SEE IF THIS IS OUR TEXT UNIT
FINDT500  DS    0H
          TM    DATUFLG,DATUDST         IS THIS A 'TO' TEXT UNIT?
          BZ    FINDT100                NO, GET NEXT TEXT UNIT
FINDT600  DS    0H
          CLM   R4,B'0011',DATUKEY      IS THIS THE ONE WE WANT?
          BNE   FINDT100                NO, GET THE NEXT TEXT UNIT
          TM    DATUFLG,DATUNEG         IS THIS A NEGATED KEYWORD?
          BNZ   FINDT100                YES
FINDT700  DS    0H
          SLR   R15,R15                 SET R15 TO '0'. TEXT UNIT FOUND
          B     FINDT900                GO TO NORMAL RETURN
FINDT800  DS    0H
          SLR   R1,R1                   SET TEXT UNIT POINTER REG TO 0
          LA    R15,4                   SET R15 TO '04'. TEXT UNIT NOT
                                        FOUND
FINDT900  DS    0H
          L     R14,SAVAREA2+12         RESTORE CALLER'S REGISTER 14
          L     R0,SAVAREA2+20          RESTORE CALLER'S REGISTER 0
          LM    R2,R12,SAVAREA2+28      RESTORE REGISTER 2 -- REGISTER 12
          BR    R14
```

Appendix G. Accessibility

Accessible publications for this product are offered through IBM Documentation (www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact the z/OS team web page (www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Documentation with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1)

are mutually exclusive alternatives. If you hear the lines `3.1 USERID` and `3.1 SYSTEMID`, your syntax can include either `USERID` or `SYSTEMID`, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The `*` symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element `*FILE` with dotted decimal number 3 is given the format `3 \* FILE`. Format `3* FILE` indicates that syntax element `FILE` repeats. Format `3* \* FILE` indicates that syntax element `* FILE` repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines `5.1*, 5.1 LASTRUN, and 5.1 DELETE` mean that if you use more than one of the `LASTRUN` and `DELETE` syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line `2.1 %OP1` means that you must refer to separate syntax fragment `OP1`.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example `5? NOTIFY`). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines `5 ?, 5 NOTIFY`, and `5 UPDATE`, you know that the syntax elements `NOTIFY` and `UPDATE` are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the `!` symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines `2? FILE, 2.1! (KEEP), and 2.1 (DELETE)`, you know that `(KEEP)` is the default option for the `FILE` keyword. In the example, if you include the `FILE` keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the `FILE` keyword is omitted, the default `FILE(KEEP)` is used. However, if you hear the lines `2? FILE, 2.1, 2.1.1! (KEEP)`, and `2.1.1 (DELETE)`, the default option KEEP applies only to the next higher dotted decimal number, `2.1` (which does not have an associated keyword), and does not apply to `2? FILE`. Nothing is used if the keyword `FILE` is omitted.

*** indicates an optional syntax element that is repeatable**
The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line `5.1* data area`, you know that you can include one data area, more than one data area, or no data area. If you hear the lines `3* , 3 HOST, 3 STATE`, you know that you can include `HOST, STATE`, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write `HOST STATE`, but you cannot write `HOST HOST`.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line `6.1+ data area`, you must include at least one data area. If you hear the lines `2+, 2 HOST,` and `2 STATE`, you know that you must include `HOST, STATE,` or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com®/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming Interface Information

This book is intended to help the customer install BDT.

This book also documents intended Programming interfaces that allow the customer to write programs to obtain the services of z/OS BDT. This information is identified where it occurs by an introductory statement to a chapter.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

GLOSSARY

This glossary defines important terms and abbreviations used in this book. If you do not find the term you are looking for, refer to the index or to the *IBM Dictionary of Computing* New York: McGraw-Hill, 1994.

catalog
The collection of all data set indexes that is used by the control program to locate a volume containing a specific data set.

checkpoint data set
See TQI checkpoint data set.

DAP
Dynamic application program.

dependent transaction control (DTC)
A method of controlling the scheduling of file-to-file transactions by organizing the transactions into a network in which some transactions wait for the completion of other transactions before being scheduled.

DTC
Dependent transaction control.

dynamic application program (DAP)
A part of BDT that performs a particular function, especially the transfer of data.

generic master job definition library
In BDT, a data set that contains predefined transaction definitions.

global node
In BDT, the node that schedules and manages all file-to-file transactions involving itself and a local node and responds to commands issued against those transactions.

GMJD
Generic master job definition.

Interactive System Productivity Facility (ISPF)
A licensed program that provides menus and data entry panels for using system functions.

ISPF
Interactive System Productivity Facility.

JES3
Job entry subsystem 3.

job entry subsystem 3 (JES3)
A component of MVS that receives jobs into the system and processes all output data produced by the jobs. JES3 exerts centralized control over multiple processor complexes.

local node
In BDT, the node that receives file-to-file transactions and commands submitted by users and sends them to the global node for processing.

MCS
Multiple console support.

multiple console support (MCS)
A feature of MVS that permits selective message routing to multiple operator's consoles.

network
In BDT, two or more BDT nodes that are joined by SNA sessions.

network job entry (NJE)
The transmission of jobs, in-stream data sets, operator commands and messages, system output data sets, and job accounting information from one computer complex to another across a telecommunication link. Synonymous with *job networking*.

NJE
Network job entry.

node
In BDT, the point in a BDT address space that is linked to another BDT address space for either file-to-file communication or SNA NJE communication.

poly-BDT complex
A JES complex that has more than one BDT address space.

RACF
Resource Access Control Facility.

Resource Access Control Facility (RACF)
A licensed program that provides for access control by identifying and verifying users to the system, authorizing access to DASD data sets, logging detected unauthorized attempts to enter the system, and logging detected accesses to protected data sets.

session
In SNA, a logical connection between two network-addressable units. The connection can be activated, deactivated, or tailored to provide different protocols.

SNA
Systems Network Architecture.

Systems Network Architecture (SNA)
The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

Time sharing option (TSO)
A component of MVS that provides interactive computing from remote stations.

TQI
Transaction queuing integrity.

TQI checkpoint data set
In BDT, a data set on which the transaction queuing integrity (TQI) facility records user-submitted commands and file-to-tile transactions before sending them to the BDT address space. Should a command or transaction fail to reach the BDT address space, BDT automatically recovers it from the TQI checkpoint data set and attempts the transfer again.

transaction
In BDT, (1) a request to copy a data set, transmit a SNA NJE job, or transmit SNA NJE output (SYSOUT), and (2) the work that BDT does to process the request. Requests to copy data sets are submitted to BDT by users. Requests to transmit SNA NJE jobs and output are submitted to BDT by JES3.

transaction definition
In BDT, a character string that identifies the data set that BDT is to copy, the data set into which BDT is to write the copied data set, and parameter values that BDT is to use while copying the data set.

transaction queuing integrity
In BDT, a program that records commands and file-to-file transactions on a data set at the submitting node, thus allowing the transfers to be resubmitted automatically should they not reach the BDT work queue. TQI also allows users to receive messages.

TSO
Time sharing option.

user exit
A point in an IBM-supplied program at which a user exit routine may be given control.

user exit routine
A routine written by a user to take control at a user exit of a program supplied by IBM.

work queue
In BDT, a queue whose elements represent work that BDT must do on behalf of a transaction.

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Product Number: 5650-ZOS

SC14-7582-50