Tivoli

# Information Management for z/OS

*Guide to Integrating with Tivoli Applications*

*Version 7.1* SC31-8744-00

Tivoli

# Information Management for z/OS

*Guide to Integrating with Tivoli Applications*

*Version 7.1* SC31-8744-00

**Tivoli Information Management for z/OS Guide to Integrating with Tivoli Applications**

**Programming Interface Information**

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of Tivoli Information Management for z/OS.

# Contents

# Preface

This guide describes a number of Tivoli® applications that integrate with Tivoli Information Management for z/OS. Some of these applications, such as the NetView® AutoBridge product and the NetView Bridge Adapter, have long been associated with Tivoli Information Management for z/OS. NetView AutoBridge, for example, can be used to build and perform Tivoli Information Management for z/OS transactions in response to the NetView alerts, messages, and application data that you specify. The NetView Bridge Adapter uses the NetView Bridge function of NetView to enable NetView applications.

This guide describes the ways in which Tivoli Information Management for z/OS extends its performance potential by providing means of integrating with a wide range of Tivoli applications. An overview of these applications is described in "What This Guide Contains" on page xvi.

There may be references in this publication to versions of Tivoli Information Management for z/OS's predecessor products. For example:

- TME 10™ Information/Management Version 1.1
- Information/Management Version 6.3, Version 6.2, Version 6.1
- Tivoli Service Desk for OS/390® Version 1.2

## Who Should Read This Guide

This guide is intended for system integrators, network planners, and operators who are responsible for the installation and customization of Autobridge, the Netview Bridge Adapter, or any of the other Tivoli applications with which Tivoli Information Management for z/OS interfaces.

## Prerequisite and Related Documentation

The library for Tivoli Information Management for z/OS Version 7.1 consists of these publications. For a description of each, see "The Tivoli Information Management for z/OS Library" on page 419.

*Tivoli Information Management for z/OS Application Program Interface Guide*, SC31-8737-00

*Tivoli Information Management for z/OS Client Installation and User's Guide*, SC31-8738-00

*Tivoli Information Management for z/OS Data Reporting User's Guide*, SC31-8739-00

*Tivoli Information Management for z/OS Desktop User's Guide*, SC31-8740-00

*Tivoli Information Management for z/OS Diagnosis Guide*, GC31-8741-00

*Tivoli Information Management for z/OS Guide to Integrating with Tivoli Applications*, SC31-8744-00

*Tivoli Information Management for z/OS Integration Facility Guide*, SC31-8745-00

*Tivoli Information Management for z/OS Licensed Program Specification*, GC31-8746-00

*Tivoli Information Management for z/OS Master Index, Glossary, and Bibliography*, SC31-8747-00

*Tivoli Information Management for z/OS Messages and Codes*, GC31-8748-00

*Tivoli Information Management for z/OS Operation and Maintenance Reference*, SC31-8749-00

*Tivoli Information Management for z/OS Panel Modification Facility Guide*, SC31-8750-00

*Tivoli Information Management for z/OS Planning and Installation Guide and Reference*, GC31-8751-00

*Tivoli Information Management for z/OS Problem, Change, and Configuration Management*, SC31-8752-00

*Tivoli Information Management for z/OS Program Administration Guide and Reference*, SC31-8753-00

*Tivoli Information Management for z/OS Reference Summary*, SC31-8754-00

*Tivoli Information Management for z/OS Terminal Simulator Guide and Reference*, SC31-8755-00

*Tivoli Information Management for z/OS User's Guide* , SC31-8756-00

*Tivoli Information Management for z/OS World Wide Web Interface Guide*, SC31-8757-00

**Note:** Tivoli is in the process of changing product names. Products referenced in this manual may still be available under their old names (for example, TME 10 Enterprise Console instead of Tivoli Enterprise Console®).

## What This Guide Contains

Information in this guide can be grouped into several topics:

- I — The NetView Bridge Adapter, contains information about the NetView Bridge Adapter; chapters within this part explain how to set up the NetView Bridge Adapter, use of transaction processors, return codes, and other features of which you should be aware.
- II — NetView AutoBridge, contains information about the NetView AutoBridge; the chapters in this part describe functional elements, data flow, format and commands, and numerous other topics.
- Problem Service provides distributed help desk applications with an interface to Tivoli Information Management for z/OS. III — Problem Service, contains information about Problem Service.
- Tivoli Information Management for z/OS has traditionally been a tool that is used in a host environment; another product, Tivoli Service Desk 6.0 (TSD), provides function similar to Tivoli Information Management for z/OS, but in a workstation environment. IV — Tivoli Service Desk Bridge, describes how these two tools work together in an interface called the Tivoli Service Desk Bridge.
- Tivoli Information Management for z/OS integrates with Tivoli's other software products in the area of Administration and Operations Management. V — Integrating with Other Tivoli Products, contains chapters on several of these products:
  - "Integrating with Tivoli Inventory" on page 369, describes Tivoli Information Management for z/OS and Tivoli Inventory. Tivoli Inventory is a hardware and

software inventory-gathering application designed to help system administrators and accounting personnel manage the complexity of PC and UNIX® systems in a distributed client/server enterprise.

- “Integrating with Tivoli Enterprise Console (TEC)” on page 387, contains information on the Tivoli Enterprise Console (TEC). TEC acts as a central resource that receives information from many sources, such as systems, databases, and other applications. It integrates with major network management platforms, and collects, processes, and automatically initiates corrective actions to system, application, network, and database events.
- “Integrating with Tivoli Software Distribution” on page 395, contains information on Tivoli Software Distribution, which automates the process of distributing software to clients and servers throughout an enterprise. It allows you to install and update applications and software in a coordinated, consistent manner across platforms for timely client/server application deployment. Using Tivoli Software Distribution, you can remotely install the Tivoli Information Management for z/OS HLAPI client programs. You can also have Tivoli Information Management for z/OS change requests initiate the distribution of workstation software.
- “Tivoli Decision Support” on page 409 describes where to find information about using Tivoli Decision Support with Tivoli Information Management for z/OS data.

## How Information Is Presented in This Guide

The panels presented in this guide are not meant to be exact replicas of the way a panel appears on the screen. The information on the panels is correct, but the spacing is not always exact.

In the text of this book, selections on selection or options panels and fields on data-entry panels appear in bold type; for example, the **Return to caller** field. The input you enter in response to the fields on data-entry panels is in all capital letters; for example, Enter CREATE in the **Create/inquiry** field.

Commands, such as END, CONTROL, RESUME, or FIELD, appear in all capital letters in text. Although not commands, the user responses YES and NO also appear in capital letters.

The highlighted print on a panel indicates the selection you are to make; the highlighted print in text is the information you enter or select while performing a task.

## Contacting Customer Support

For support inside the United States, for this or any other Tivoli product, contact Tivoli Customer Support in one of the following ways:

- Send e-mail to **support@tivoli.com**
- Call 1-800-TIVOLI8
- Navigate our Web site at **http://www.support.tivoli.com**

For support outside the United States, refer to your Customer Support Handbook for phone numbers in your country. The Customer Support Handbook is available online at **http://www.support.tivoli.com**.

When you contact Tivoli Customer Support, be prepared to provide identification information for your company so that support personnel can assist you more readily.

The latest downloads and fixes can be obtained at **http://www.tivoli.com/infoman**.

# I — The NetView Bridge Adapter

# 1 Introducing the NetView Bridge Adapter

Systems management involves both operational activities, such as monitoring and control, and administrative activities, such as tracking and reporting. Until the development of the NetView Bridge and the Tivoli Information Management for z/OS NetView Bridge Adapter, these two types of activities were supported by separate products that could not be easily interconnected under program control. Coordinating these activities required human intervention or expensive additional programming.

For example, when a system component detected a problem and reported it to NetView, NetView performed its diagnosis and resolution process for that problem. Tivoli Information Management for z/OS provided a function for tracking the problem, but there was no simple, automatic way for NetView to enter the problem into the Tivoli Information Management for z/OS database. Also, real-time activities performed by NetView automation routines frequently needed access to configuration data maintained by Tivoli Information Management for z/OS. Once again, there was no automated way to access this data from NetView. The NetView Bridge Adapter and NetView Bridge integrate these functions into the NetView automation platform.

The Tivoli Information Management for z/OS NetView Bridge Adapter enables the NetView and Tivoli Information Management for z/OS products to work together. Along with the NetView Bridge, it enables automated message-handling functions. These functions consist of message routing and transmission within the NetView address space (the NetView Bridge), and message processing and submission to the Tivoli Information Management for z/OS High-Level Application Program Interface (HLAPI). The HLAPI, in turn, interfaces with the Tivoli Information Management for z/OS Low-Level API (LLAPI), which accesses the database.

The Adapter provides the connection between the NetView Bridge and the HLAPI. It transforms user-written NetView automation command procedures or server requests into HLAPI transactions and responses. Through the NetView Bridge Adapter, you can use NetView to:

- Create records in a Tivoli Information Management for z/OS database
- Update records in a Tivoli Information Management for z/OS database in a manner that protects the integrity of the records
- Retrieve a list of records or a single record from a Tivoli Information Management for z/OS database based upon a set of search criteria
- Perform user-defined tasks on records in a Tivoli Information Management for z/OS database

For more information on the NetView Bridge, refer to the *NetView Bridge Implementation* manual. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the HLAPI.

## Software Required for the NetView Bridge Adapter?

The software requirements for the NetView Bridge Adapter are:

- Tivoli Information Management for z/OS (program number 5697-SD9)
- NetView Version 3 for MVS/ESA™ (5655–007) or a subsequent release
- OS/PL1 Library, Version 2 Release 3 (5668-911) or a subsequent release; if you are using NetView for OS/390 Version 1 Release 1 (5697–B82) or later, you must use Language Environment® (available with OS/390 and z/OS).

## How Does the NetView Bridge Adapter Work?

The NetView Bridge Adapter consists of the server address space and transaction processors. The transaction processors enable you to manipulate the data in your Tivoli Information Management for z/OS database. You can define your own transaction processors to perform tasks not provided by the NetView transaction processors.

To start the NetView Bridge Adapter, you must initialize NetView, the NetView Bridge Adapter server address space, and the Tivoli Information Management for z/OS product. First you must initialize the NetView Bridge dispatcher autotask that resides in NetView. To see how to do this, refer to the *NetView Bridge Implementation* manual. Next, you must initialize the adapter. The adapter automatically initializes Tivoli Information Management for z/OS after you initialize the adapter. One way to start initialization is with a JCL EXEC called as part of NetView startup. A sample of this JCL appears as procedure BLGBSPCX in the data set SBLMSAMP.

The adapter supports this sequential startup by interfacing with the NetView Bridge server support API on the NetView side of the process, and by interfacing with the HLAPI on the Tivoli Information Management for z/OS side of the process. For more details about the NetView Bridge server support API, refer to the *NetView Bridge Implementation* manual. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the HLAPI.

Figure 1 on page 5 shows the interaction between a user-written command, the NetView Bridge, the adapter, and Tivoli Information Management for z/OS. The NetView Bridge consists of the Bridge Requester API, the Server Support API, and the bridge dispatcher.

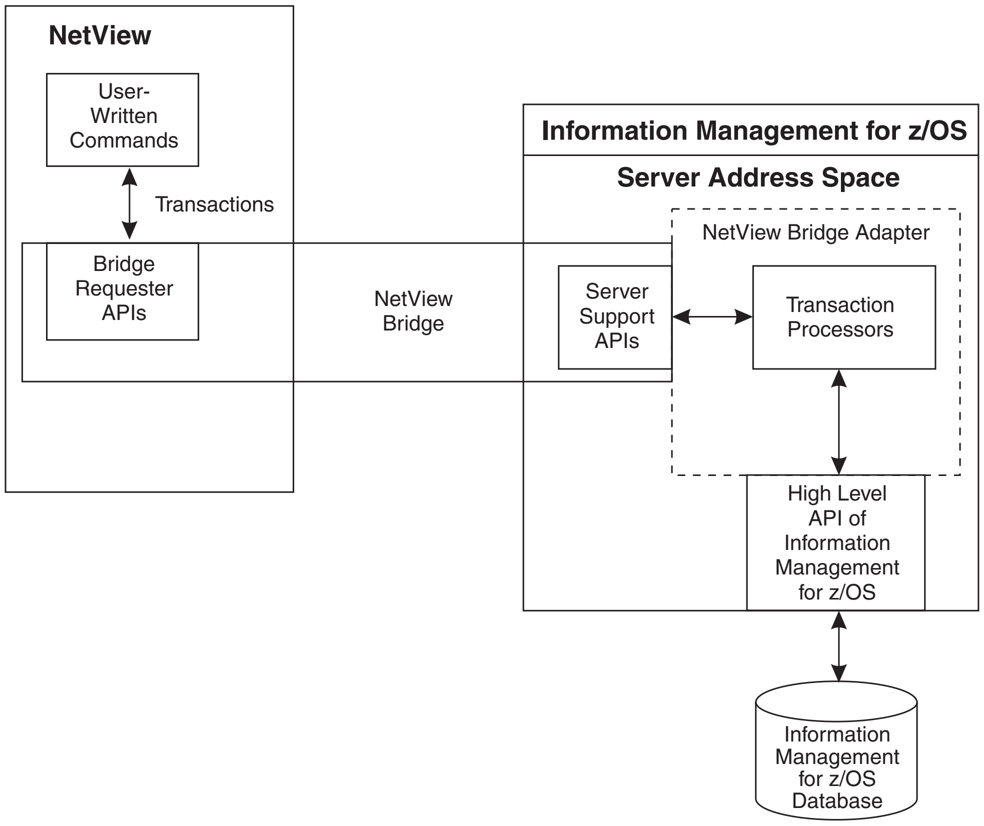


*Figure 1. Tivoli Information Management for z/OS NetView Bridge Adapter*

The NetView Bridge Adapter acts as the connection between the commands you write in NetView and the database you use in Tivoli Information Management for z/OS. The transaction specified in your NetView command is processed as follows:

1. A user-written command procedure (NetView Command List Language, Restructured Extended Executor (REXX), or high-level language (HLL) program) sends the transaction to the NetView Bridge Adapter server address space (also called the database server) using the NetView Bridge Requester API.
2. The bridge dispatcher (a NetView autotask) puts this request on a NetView program-to-program interface (PPI) queue to access the first available database server.
3. When the database server receives a request on its PPI queue, it uses the server support API to retrieve the transaction and start the appropriate transaction processor.
4. The transaction processor starts the HLAPI to perform the function you request.
5. If this request asks the transaction processor for a reply, the processor calls the server support API with all the data for the reply. The NetView Bridge transmits the transaction reply back to the user-written command procedure, which uses the NetView Bridge Requester API to extract the data for you.

# 2

# Preparing to Use the NetView Bridge Adapter

The NetView Bridge Adapter supports automated operations and therefore does not provide user-interactive panels, commands, or dialogs. The adapter consists of two basic parts: the NetView Bridge Adapter address space and the transaction processors. For more detailed information about the transactions, see "Performing the NetView Bridge Adapter Transactions" on page 21. The adapter address space provides access to the Tivoli Information Management for z/OS database, manages the interface with NetView using the NetView Bridge, and provides the job control process for running under MVS™. The events that occur during an Adapter session are:

- Initializing the adapter address space
- Controlling server processing
- Controlling transaction processing
- Recording transaction requests and replies
- Closing the NetView Bridge Adapter address space

Before any of the above can happen, you must install the NetView Bridge Adapter modules. This can be done when the Tivoli Information Management for z/OS program product is installed or later. You must then make the NetView Bridge Adapter available to the transactions and command procedures that use it.

## Installing the NetView Bridge Adapter

To install the NetView Bridge Adapter, you will use SMP/E, described in the *Program Directory* for Tivoli Information Management for z/OS.

## Setting Up the NetView Bridge Adapter

To make the NetView Bridge Adapter available, create the database server by following these steps:

1. Identify the NetView installation that is to use the NetView Bridge Adapter. Tivoli Information Management for z/OS must be installed and running on this same system.
2. Set up a NetView Bridge for the NetView Bridge Adapter by following the directions in the *NetView Bridge Implementation* manual.
3. Create the data set INFOBRDS (see "The User-Supplied Input Data Set" on page 10).
4. Assign a unique program-to-program interface (PPI) receiver queue name for the server. Refer to the *NetView Application Programming Guide: Program-to-Program Interface* manual for the format of these queue names.

**Note:** The PPI receiver queue name is the RCVQUEUE parameter of the JCL procedure. See "Modifying the JCL Supplied with the NetView Bridge Adapter".

5. Determine the application identifier (APPLID) for the server address space. The APPLID is used as a high-level qualifier for any data set the NetView Bridge Adapter will dynamically allocate and is the ID that must be in the INITCLAS parameter in INFOBRDS. For more information, see "The EXEC Statement" on page 9.
6. Define automation logic that issues the MVS START command to start the NetView Bridge Adapter address space. Modify the JCL procedure supplied with this product to include the APPLID name and the PPI receiver queue name (JCL parameter RCVQUEUE) you defined in steps 4 on page 7 and 5 above.
7. Allocate the output data sets that you reference in the modified JCL (see "The User-Supplied Output Data Sets" on page 17).

**Note:** You will need one procedure for each Bridge Adapter that you want to use.

## Modifying the JCL Supplied with the NetView Bridge Adapter

The Tivoli Information Management for z/OS NetView Bridge Adapter provides a sample startup JCL procedure BLGBSPCX in the data set SBLMSAMP. This section explains what to do with it to get the NetView Bridge Adapter working.

### Defining the JCL Procedure

This is an example of a typical JCL startup procedure for a Tivoli Information Management for z/OS NetView Bridge Adapter server address space. The example on page 11 can be used as a model when creating data sets. You must modify the JCL EXEC that is supplied with the adapter to fit your particular user environment. An explanation of each part of the procedure shown in **bold-faced** type follows the example.

```
//*-----------------------------------------------------------------* 00010000
//*                                                                 * 00020000
//*             Licensed Materials - Property of IBM                * 00030000
//*                                                                 * 00040000
//*                          5697-SD9                               * 00050000
//*                                                                 * 00060000
//*           (C) Copyright IBM Corporation, 1981, 2001.            * 00070000
//*                                                                 * 00080000
//*                  See Copyright Instructions                     * 00090000
//*                                                                 * 00100000
//*-----------------------------------------------------------------* 00110000
//*****************************************************************  00120000
//*                                                               *  00130000
//* NAME: BRIDGEPR (NETVIEW BRIDGE ADAPTER STARTUP PROCEDURE)       *  00140000
//*                                                               *  00150000
//* PURPOSE: START A NETVIEW BRIDGE ADAPTER SERVER ADDRESS SPACE    *  00160000
//*                                                               *  00170000
//*****************************************************************  00180000
//BRIDGEPR PROC                                                      00190000
//*                                                                  00200000
//* THE PARAMETERS ON THE NEXT LINE ARE PASSED TO THE ADAPTER.       00210000
//SERVER EXEC PGM=BLGBSPC,PARM='/APPLID,RCVQUEUE',REGION=6M          00220000
//*                                                                  00230000
//* THE DATA SET NAME QUALIFIER, LOCAL, IMPLIES THAT WHERE THE       00240000
//* QUALIFIER IS USED IT IS UNIQUE TO AN ADDRESS SPACE INSTANCE.     00250000
//*                                                                  00260000
//* THE STEPLIB IS USED IF THE INFORMATION MANAGEMENT FOR Z/OS CODE  00270000
//* AND THE PL/I OR THE LE RUN TIME LIBRARIES ARE NOT IN LINKLIST.   00280000
//* TO USE LE YOU MUST HAVE THE CORRECT VERSION OF NETVIEW.          00290000
//* ALL STEPLIB DATASETS MUST BE AUTHORIZED                          00300000
//*                                                                     00310000
//STEPLIB DD  DISP=SHR,DSN=BLM.SBLMMOD1                                 00320000
//      DD  DISP=SHR,DSN=SYS1.NETVIEW.CNMLINK                           00330000
//* USE CEE.SCEERUN IN PLACE OF PLILINK AND SIBMLINK IF YOU ARE USING LE00340000
//      DD  DISP=SHR,DSN=SYS1.PLI.PLILINK                               00350000
//      DD  DISP=SHR,DSN=SYS1.PLI.SIBMLINK                              00360000
//INFOBRDS  DD DISP=SHR,DSN=SYSTEM.BRIDGE.INFOBRDS                      00370000
//IBRPRINT  DD DISP=SHR,DSN=LOCAL.SERVER.IBRPRINT,                      00380000
//  DCB=(RECFM=F,LRECL=104,BLKSIZE=104)                                 00390000
//HLAPILOG  DD DISP=SHR,DSN=LOCAL.SERVER.HLAPILOG                       00400000
//APIPRINT  DD DISP=SHR,DSN=LOCAL.SERVER.APIPRINT                       00410000
//SYSPRINT  DD DISP=SHR,DSN=LOCAL.SERVER.SYSPRINT                       00420000
//SYSUDUMP  DD DISP=SHR,DSN=LOCAL.SERVER.SYSUDUMP                       00430000
```

In the preceding JCL example, the high-level qualifier LOCAL implies that these items are unique for each setup of the NetView Bridge Adapter. The values for the procedure name, APPLID, and receiver queue name, RCVQUEUE, must also be unique for each adapter server address space. You can use the JCL in this example to start multiple copies of the adapter as long as you modify the JCL to contain these unique parameters each time. See "Creating Additional Copies of a Server" on page 18 for more information.

**Notes:**

1. The only data sets that you need to have in your STEPLIB are CNMLINK and SBLMMOD1. The others are probably in LINKLIST.
2. The STEPLIB, INFOBRDS, IBRPRINT, and SYSPRINT DD statements are required. HLAPILOG, SYSUDUMP, and APIPRINT are optional. To allocate these data sets, see "The User-Supplied Output Data Sets" on page 17.

## The JCL Procedure Parameters

The JCL procedure supplied with the NetView Bridge Adapter consists of three parts: the EXEC statement, the STEPLIB concatenation, and the user-supplied data sets. Each bold-faced word in the following section matches one in the JCL example.

### The EXEC Statement

The **EXEC** statement of this JCL supplies several transaction input parameters. The two EXEC parameters listed below must be unique for each copy of the server, and they must be entered in this *exact* format:

```
'/applid,rcvqueue'
```

The slash (/) symbol differentiates between PL/I run-time parameters and NetView Bridge Adapter parameters. Any parameter before the slash is a PL/I run-time parameter, and any parameter after the slash is a NetView Bridge Adapter parameter. In this JCL procedure, both APPLID and RCVQUEUE are NetView Bridge Adapter parameters. If the slash is omitted, the system assumes that all parameters are NetView Bridge Adapter parameters.

**APPLID**
: The unique application identifier used with Tivoli Information Management for z/OS for this server address space. The value specified for APPLID is used within Tivoli Information Management for z/OS privilege class processing. This entails adding this APPLID as an eligible user of a specific privilege class record in the Tivoli Information Management for z/OS database. The APPLID must be an eligible user of the privilege class specified in the initclass field (see 12). It must also be a valid MVS high-level qualifier for data set names on your system. It can be 1 to 8 characters long. The first character must be alphanumeric. APPLID correcsponds to

the APPLICATION_ID PDB used by the HLAPI. For more information about APPLICATION_ID, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* .

**RCVQUEUE**
The unique NetView PPI receiver queue name identifier used by this address space to receive incoming transactions. The identifier is 1 to 8 characters long and can contain uppercase alphabetic (A–Z), numeric (0–9), and these special characters: $, %, &, @, and #. For more information, refer to the *NetView Application Programming Guide: Program-to-Program Interface*.

**REGION**
The REGION size listed in this JCL procedure is 6 megabytes. This size is best under conditions of heavy loading. You may lower this number to 4 megabytes if you do not experience *out of memory* abnormal end (abend) conditions, such as TSD/390 user abend 804, at the lower value.

### The STEPLIB Concatenation

The **STEPLIB** statement defines the location of the NetView Bridge Adapter load modules, the Tivoli Information Management for z/OS load modules, and the NetView load modules. It also specifies the concatenation order that they are to be searched in while performing the load.

### The User-Supplied Input Data Set

INFOBRDS is the DDNAME for the data set containing the parameters used to initialize the NetView Bridge Adapter server address space. Allocate this data set using fixed blocked record format (RECFM=FB) physical sequential organization (DSORG=PS), a record length of 80, and a block size which is a multiple of 80. This data set must not be a SYSOUT file. It also contains some of the parameters used to initialize the Tivoli Information Management for z/OS product. This is the only user-supplied data set into which you must put data when you create it. A single copy of this data set can be shared by all of the adapter address spaces. You can use the sample job EYLSJ002, supplied in SBLMSAMP, to allocate the data sets for use by your procedure.

```
//EYLSJ002 JOB 'your-job-card'
//*-----------------------------------------------------------------*
//*                                                                 *
//*            Licensed Materials - Property of IBM.                *
//*                                                                 *
//*                           5697-SD9                              *
//*                                                                 *
//*           (C) Copyright IBM Corporation, 1981, 2001.            *
//*                                                                 *
//*                   See Copyright Instructions                    *
//*                                                                 *
//*-----------------------------------------------------------------*
//*******************************************************************
//*  THIS JCL WILL ALLOCATE THE NETVIEW BRIDGE ADAPTER
//*  DATA SETS ALONG WITH THE POSTPROCESSOR DATA SETS.
//*
//*  MODIFY ALL OCCURRENCES OF THE FOLLOWING WITH UPPERCASE TEXT:
//*
//*  1) 'your-job-card' TO SUIT YOUR SYSTEM REQUIREMENTS
//*  2) 'tvol' TO THE VOLUME TARGET DATA SETS WILL BE CREATED ON
//*  3) ANY OTHER PROC PARAMETERS, IF THE SUPPLIED
//*       DEFAULT IS NOT ACCEPTABLE
//*
//*  IF YOU DO NOT PLAN TO RUN THE POSTPROCESSOR THEN COMMENT OUT
//*  OR DELETE THE //SYSTSIN AND //SYSTSPRT DATA SETS.
//*
```

```
//*********************************************************************
//ALLOC  PROC A='*',      **  PRINT SYSOUT DEFAULT
//       BLMPRFX=EYL,     **  PRE-QUALIFIER FOR DATA SETS
//       BLMVER=BRIDGE,   **  2ND-QUALIFIER FOR DATA SETS
//       DSP=CATLG,       **  DATA SET DISPOSITION
//       TGVOL=tvol,      **  VOLSER FOR DATA SETS
//       TGUNIT=tunit     **  UNIT TYPE FOR DATA SETS
//*********************************************************************
//PSTEP1   EXEC PGM=IEFBR14
//*
//*  NETVIEW BRIDGE ADAPTER DATA SETS
//*
//SYSPRINT DD SYSOUT=&A
//HLAPILOG DD DSN=&BLMPRFX..&BLMVER..HLAPILOG,
//            SPACE=(CYL,(1,1)),
//            DCB=(RECFM=VBA,DSORG=PS,LRECL=125,BLKSIZE=6144),
//            UNIT=&TGUNIT,VOL=SER=&TGVOL,DISP=(,&DSP)
//INFOBRDS DD DSN=&BLMPRFX..&BLMVER..INFOBRDS,
//            SPACE=(TRK,(1,1)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200,DSORG=PS),
//            UNIT=&TGUNIT,VOL=SER=&TGVOL,DISP=(,&DSP)
//APIPRINT DD DSN=&BLMPRFX..&BLMVER..APIPRINT,
//            SPACE=(CYL,(1,1)),
//            DCB=(RECFM=VBA,DSORG=PS,LRECL=125,BLKSIZE=6144),
//            UNIT=&TGUNIT,VOL=SER=&TGVOL,DISP=(,&DSP)
//IBRPRINT DD DSN=&BLMPRFX..&BLMVER..IBRPRINT,
//            SPACE=(TRK,(10,1)),
//            DCB=(RECFM=F,DSORG=PS,LRECL=104,BLKSIZE=104),
//            UNIT=&TGUNIT,VOL=SER=&TGVOL,DISP=(,&DSP)
//SYSTSIN  DD DSN=&BLMPRFX..&BLMVER..SYSTSIN,
//            SPACE=(3200,1),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200,DSORG=PS),
//            UNIT=&TGUNIT,VOL=SER=&TGVOL,DISP=(,&DSP)
//SYSTSPRT DD DSN=&BLMPRFX..&BLMVER..SYSTSPRT,
//            SPACE=(CYL,(1,1)),
//            DCB=(RECFM=VBA,LRECL=125,BLKSIZE=1632,DSORG=PS),
//            UNIT=&TGUNIT,VOL=SER=&TGVOL,DISP=(,&DSP)
//PROCEND  PEND
//J1ALC  EXEC ALLOC
/*
```

Set up this data set with one line per parameter. Use the following values when you allocate this data set:

- LRECL=80
- RECFM=FB
- DSORG=PS

The format of the INFOBRDS parameters is shown in this data set.

```
SEND_QUEUE='send_queue'
,INITCLAS='initclas'
,SESSMEMB='sessmemb'
[,CLASS_COUNT=class_count]
,API_MSG='api_msg'
,HLI_MSG='hli_msg'
[,TIMEOUT_INTERVAL=nnn]
[,SPOOL_INTERVAL=spool_interval]
[,ALIAS_TABLE_CNT=alias_table_cnt]
,DATABASE_ID='database_id'
,DEFAULT_OPTION='default_option'
[,DEFAULT_STORAGE=default_storage]
,IBRPRINT_OPTION='ibrprint_option'
[,TIINQLIM=tiinqlim]
,TRANSLATE='translate'
,VALIDATE='validate'
,BYPASS_PANEL_PROCESSING='YES'|'NO'
```

```
,TRANSACTION_PROCESSOR='program_name'
,USE_DATE_VIEW='YES'|'YES'
,PRELOAD_TRAN_PROC='YES'|'NO'
,DATE_CONVERSION='YES'|'NO';
```

This data set is a stream file that is read using a PL/I data-directed GET, and all rules applying to the layout of data-directed input apply to this data set. If you edit this data set with the ISPF/PDF editor, be sure that you do not have sequence numbering turned on. Allocate this data set as fixed block (FB) with a record length of 80. Refer to the *OS PL/I Programming: Language Reference* for information about PL/I routines. You must enter the capitalized names exactly as shown, and then follow them with an equal sign. The parameters shown in brackets [ ] denote optional parameters. The file must end with a semicolon. A description of each field follows the example.

**Note:** The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information concerning the parameters SESSMEMB through DEFAULT_STORAGE.

The descriptions of these fields are:

**SEND_QUEUE**
This is a required field. You must enter its value in single quotation marks because it is a text field. It specifies the PPI to send the queue name used to send transactions to NetView. This field must match the *oqueue* parameter of the NetView Bridge RTRINIT command. For further information, refer to the *NetView Bridge Implementation* manual.

**INITCLAS**
This is a required field. You must enter its value in single quotation marks because it is a text field. It is the privilege class name that the adapter servers use to start Tivoli Information Management for z/OS and serves as the default privilege class. The privilege class specified here must contain an eligible user ID that is equal to the value specified for APPLID (see page 9).

If the initialization privilege class named in this field gives you authority to perform the transaction that you want, then you need not specify a separate privilege class field in the transaction parameters that you send from your NetView command.

If the initialization privilege class does not provide the authority needed to perform the transaction, then you must specify a different privilege class in the adapter transaction parameters. The parameters CREPRIV, INQPRIV, RETRPRIV, or UPDPRIV enable you to use a different privilege class. See Table 5 on page 29, Table 7 on page 32, and Table 8 on page 35 for more information about these variables. The *Tivoli Information Management for z/OS Program Administration Guide and Reference* contains additional information about privilege classes.

**SESSMEMB**
This is a required field. You must enter its value in single quotation marks because it is a text field. It contains the name of the Tivoli Information Management for z/OS session-parameter member. Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for more information.

**CLASS_COUNT**
This is an optional field. It indicates the number of Tivoli Information Management for z/OS privilege class records that can be maintained in storage at one time during this Tivoli Information Management for z/OS NetView Bridge Adapter session. This

saves retrieval time during the processing of transactions. Once the limit is met, a new class record replaces the oldest class record in storage. If you do not indicate a value, the Tivoli Information Management for z/OS session operates with one class record in storage at a time.

**API_MSG**
This is an optional field. You must enter its value in single quotation marks because it is a text field. The valid entries for this field are:

**P** Specifies that Tivoli Information Management for z/OS LLAPI messages are written to the APIPRINT data set.

**C** Specifies that Tivoli Information Management for z/OS LLAPI messages are returned to the Tivoli Information Management for z/OS HLAPI.

**B** Specifies that Tivoli Information Management for z/OS LLAPI messages are both written to the APIPRINT data set and returned to the Tivoli Information Management for z/OS HLAPI.

If you specify any character other than those above or if you leave this field blank, then option C is the default.

**HLI_MSG**
This is an optional field. You must enter its value in single quotation marks because it is a text field. The valid entries for this field are:

**P** Specifies that Tivoli Information Management for z/OS HLAPI messages are written to the HLAPILOG data set.

**C** Specifies that Tivoli Information Management for z/OS HLAPI messages are returned with each transaction reply to the requester in the *messages* transaction parameter.

**B** Specifies that Tivoli Information Management for z/OS HLAPI messages are both written to the HLAPILOG data set and returned from the Tivoli Information Management for z/OS HLAPI.

If you specify any character other than those above or if you leave this field blank, then option C is the default.

**TIMEOUT_INTERVAL**
This is an optional field. It specifies the interval of time in seconds that a Tivoli Information Management for z/OS HLAPI transaction can run before a timer interrupt occurs. If a timeout occurs, the HLAPI ends its LLAPI session and ends the HLAPI session. You must perform another HL01 transaction before you can perform additional transactions. If you omit this parameter, the HLAPI defaults to 300 seconds (5 minutes). If you specify an interval greater than 0, but less than 45 seconds, the interval is set to 45 seconds.

**SPOOL_INTERVAL**
This is an optional field. It specifies the interval of time in minutes that the Tivoli Information Management for z/OS activity logs (APIPRINT and HLAPILOG) are spooled when messages (see API_MSG, page 13 and HLI_MSG, page 13) are logged. After this interval passes, rollover occurs, and the activity logs are closed and reopened. New log information is written into the now-empty logs. If you do not specify this parameter, the Tivoli Information Management for z/OS HLAPI does *not* log messages.

On a rollover, the IBRPRINT data set, which is not controlled by the spool interval, is recycled and new log information is written starting at the top of the data set, writing over any existing information, one record at a time. At initialization time, the IBRPRINT data set is reformatted.

**ALIAS_TABLE_CNT**
This is an optional field. It defines the number of alias tables you can maintain in storage for this Tivoli Information Management for z/OS session at one time. This saves retrieval time during the processing of transactions. Once the limit is reached, a new alias table replaces the oldest alias table in storage. Any number between 0 and 256 is valid for this field. If you omit the value or enter zero, no alias table processing is done for any Tivoli Information Management for z/OS HLAPI transactions.

**DATABASE_ID**
This is an optional field. You must enter its value in single quotation marks because it is a text field. It specifies the current database's ID. For Tivoli Information Management for z/OS records this number is 5. If you do not specify a value, or you enter a value that is not valid, this field's value defaults to 5.

**DEFAULT_OPTION**
This is an optional field. You must enter its value in single quotation marks because it is a text field. It specifies whether the Tivoli Information Management for z/OS HLAPI is to perform default data response processing in this session.

When you create a record, default data response processing substitutes a default response for a missing item whenever a transaction does not find a response item in the set of input parameters. When alias table processing is specified, it finds the default responses in the alias table. Valid entries for this field are:

| | |
|---|---|
| **ALL** | The HLAPI performs default response processing for all fields. |
| **REQUIRED** | The HLAPI performs default response processing for required fields only. |
| **NONE** | The HLAPI performs no default response processing. This is the default value for this field. |

**DEFAULT_STORAGE**
This is an optional field. It specifies how much additional storage in bytes is to be allocated to hold default response data when you create a record. The default value of this parameter is 1024 bytes. The default responses are supplied by an alias table. For more information on storage, refer to the description of the DEFAULT_DATA_STORAGE_SIZE in the "Parameter Data Definition for PDB" section of the *Tivoli Information Management for z/OS Application Program Interface Guide*.

**IBRPRINT_OPTION**
This is an optional field. You must enter its value in single quotation marks because it is a text field. The valid entries for this field are:

| | |
|---|---|
| **ALL** | Specifies that the transaction logs all errors, requests, and replies in the IBRPRINT data set. |
| **ERROR** | Specifies that the transaction logs errors only. |
| **NONE** | Specifies that the transaction logs nothing in IBRPRINT. NONE is the default value. |

**DEBUG** Specifies that all NetView Bridge input parameters are to be written to the IBRPRINT data set and data tracing is to be performed by Tivoli Information Management for z/OS APIs.

**Notes:**

1. Use the DEBUG option only when you perform debugging procedures or when you are reporting problems to the support center.
2. For more information about what is written to the IBRPRINT data set, see "The NetView Bridge Adapter IBRPRINT Data Set" on page 53.

**TIINQLIM**
This is an optional field. It is a numeric field that specifies the number of transaction entries that can be simultaneously queued to this server's PPI receiver queue. The minimum value is 5 and the maximum value is 2000. The default value of this field is 5.

**TRANSLATE**
This is an optional field. You must enter its value in single quotation marks because it is a text field. Its valid values are YES and NO. It specifies if the HLAPI Parameter Data Block field contents are automatically translated from lowercase to uppercase. If you do not explicitly set this field to NO, it defaults to YES.

**VALIDATE**
This is an optional field. You must enter its value in single quotation marks because it is a text field. Its valid values are YES and NO. It determines if the Tivoli Information Management for z/OS HLAPI performs data response validation when processing input parameter data blocks (PDBs). If you do not explicitly set this field to NO, it defaults to YES.

**Note:** Bypassing Tivoli Information Management for z/OS's validation process can cause unexpected records to appear in the Tivoli Information Management for z/OS database. With VALIDATE set to NO it is possible to enter nondisplayable characters into a Tivoli Information Management for z/OS field, which can prevent interactive updates of the field. Or, if you enter two words in a single word field, you can prevent verification using the VERIFIER parameter during an update transaction, possibly causing you to update the wrong records.

**BYPASS_PANEL_PROCESSING**
This is an optional field. Specification of YES allows data view records to be used with the NetView Bridge Adapter. Valid values are YES and NO. The default value is NO. If YES is specified, you must use data view records for all transactions performed by the NetView Bridge Adapter. If NO is specified (or no value is specified), data view records can still be used instead of PIDTs if

- The INFOBRDS parameter USE_DATA_VIEW is set to YES

  ***or***

- The input data contains a data item with the name of USE_DATA_VIEW with a value of YES. The input data item USE_DATA_VIEW will override the INFOBRDS keyword parameter USE_DATA_VIEW if the INFOBRDS keyword parameter BYPASS_PANEL_PROCESSING is not set to YES.

**TRANSACTION_PROCESSOR**
This is an optional field. You must enter its value in single quotation marks because it is a text field. It identifies a user-defined transaction processor routine.

**USE_DATA_VIEW**
This is an optional field. Valid values are YES and NO. Specification of YES causes the NetView Bridge Adapter to use data view records instead of PIDTs. This means that the values used with CREVIEW, INQVIEW, UPDVIEW, and RETRVIEW will be treated as data view record names instead of PIDT names. The default value is NO. If NO is specified (or no value is specified), PIDTs are used unless the input data contains an input item with the name of USE_DATA_VIEW with a value of YES. The input data item will override this INFOBRDS keyword value unless BYPASS_PANEL_PROCESSING is set to YES. If BYPASS_PANEL_PROCESSING is set to YES, then USE_DATA_VIEW is ignored.

**PRELOAD_TRAN_PROC**
This is an optional field. You must enter its value in single quotation marks because it is a text field. Its valid values are YES and NO. If you specify YES the user-defined transaction processor is loaded into storage (but not called) during initialization of the NetView Bridge Adapter, and it remains in storage until the NetView Bridge Adapter ends. The user-defined transaction processor is called when the user-defined transaction is requested. If you specify NO the user-defined transaction processor is loaded into storage and called when the user-defined transaction is requested. If you do not explicitly set this field to YES, it defaults to NO.

**DATE_CONVERSION**
This is an optional field. You must enter its value in single quotation marks because it is a text field. Its valid values are YES and NO.

- If you specify YES, the NetView Bridge Adapter will convert dates from the NetView format (MM/DD/YY) into the external date format specified on the DATECNV keyword of the SESSMEMB parameter in the INFOBRDS data set. The NetView Bridge Adapter will look at each PDB it receives. If the PDBDATA field has a length of 8, a slash / in the third position, and a slash / in the sixth position, it assumes that the data is a date. It will then attempt to convert the date to the external date form described above. If the conversion is successful, the new date is placed into the input PDB. If the conversion is not successful, the date is not altered and is passed directly to the HLAPI. When retrieving data from Tivoli Information Management for z/OS, if the external date format in use is MM/DD/YY or YY/MM/DD or DD/MM/YY, the date is converted back to the NetView format of MM/DD/YY. This parameter applies only to the IBCREATE, IBSEARCH, and IBUPDATE transactions, and does not apply to user-written transactions.
- If you specify NO, no date conversion is attempted.

This is a sample entry for the data set INFOBRDS. Note the parameters that are entered with single quotation marks around them. Because they are text fields, you must enter them exactly as shown.

```
SEND_QUEUE='OUTPUTQ',
INITCLAS='MASTER',
SESSMEMB='BLGSES00',
CLASS_COUNT=5,
API_MSG='B',
HLI_MSG='B',
TIMEOUT_INTERVAL=300,
SPOOL_INTERVAL=10,
ALIAS_TABLE_CNT=4,
DATABASE_ID='5',
DEFAULT_OPTION='ALL',
DEFAULT_STORAGE=100,
IBRPRINT_OPTION='ALL',
TIINQLIM=5,
TRANSLATE='YES',
VALIDATE='YES',
BYPASS_PANEL_PROCESSING='YES',
TRANSACTION_PROCESSOR='BUILDREL',
USE_DATA_VIEW='NO',
PRELOAD_TRAN_PROC='NO',
DATE_CONVERSION='NO';
```

## The User-Supplied Output Data Sets

When setting up your server, you must allocate the IBRPRINT and SYSPRINT output data sets. The HLAPILOG, APIPRINT, and SYSUDUMP data sets are optional. Unlike INFOBRDS, you do not need to create any information in them.

**IBRPRINT** is the DDNAME for a required data set that the NetView Bridge Adapter logging information is written into. Note the settings for IBRPRINT in the example on page 10. To ensure data integrity, each adapter requires its own copy of this data set. If the data set fills up, it overwrites older data, preserving the most current data. At initialization time, the IBRPRINT data set is reformatted. The IBRPRINT_OPTION parameter in INFOBRDS determines what gets logged in this data set. The time it takes for the NetView Bridge Adapter to initialize is directly related to the size of this data set. Define its size to be as small as possible for your purposes. It is suggested that you use 10 tracks or fewer. Allocate this data set using fixed record format (`RECFM=F`) physical sequential organization (`DSORG=PS`), a record length of 104, and a block size of 104. This data set must not be a SYSOUT file. For information concerning what is written into this data set, see “The NetView Bridge Adapter IBRPRINT Data Set” on page 53.

**HLAPILOG** is the DDNAME for an optional data set into which the Tivoli Information Management for z/OS HLAPI records information concerning its transaction activity, including type of transaction, time, and success or failure of the transaction. The setting of the INFOBRDS parameter SPOOL_INTERVAL controls logging to this data set. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the HLAPILOG data set.

**APIPRINT** is the DDNAME for an optional data set into which the Tivoli Information Management for z/OS LLAPI records its transaction information including type of transaction, time, and success or failure of the transaction. The setting of the INFOBRDS parameter SPOOL_INTERVAL controls logging to this data set. For more information about this data set, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

**SYSPRINT** is the DDNAME for a required data set that the NetView Bridge Adapter writes error information into. If an ABEND occurs during operation of the NetView Bridge Adapter or a PL/I *on condition* occurs, and you have allocated the SYSPRINT file, then the system writes a PL/I *snapshot trace* into the SYSPRINT file.

When Tivoli Information Management for z/OS writes to SYSPRINT, it formats the data using DCB information specified by the user on either a SYSPRINT DD statement (that is, LRECL or BLKSIZE) or a TSO ALLOCATE. If the user specifies an LRECL without a BLKSIZE, Tivoli Information Management for z/OS sets the BLKSIZE to:

```
    (14*LRECL) + 4
```

If the user does not specify a BLKSIZE or an LRECL, the LRECL is set to:

```
    length of output message + 4
```

and the BLKSIZE is set to:

```
    (14*LRECL) + 4
```

If the user specifies a BLKSIZE without an LRECL, the LRECL is set to the smaller of the following two statements:

```
    length of output message + 4
    BLKSIZE - 4
```

In all cases, the LRECL must be less than or equal to `BLKSIZE - 4`. Otherwise, an ABEND occurs when the data set is opened because the data attributes are inconsistent.

**SYSUDUMP** is the DDNAME for an optional data set that the system uses to store system dump information in the event that the NetView Bridge Adapter code undergoes an ABEND condition.

## Stopping a Server

There are times when you want to stop running a server address space. You may want to do this, for example, during replacement of an old server with a new one or during off-peak hours of system use. To close a server address space, issue an MVS STOP command. The format of the command is:

```
STOP jobname
```

or

```
P jobname
```

You can use the full command STOP or its abbreviation, P. The parameter *jobname* is the name of the procedure that you use to start a particular server address space. For example, if you start a server with the procedure NBAPROC1, you stop it with this command:

```
STOP NBAPROC1
```

Only use the STOP command to close a server address space. The STOP command brings the server down in an orderly manner. Do not use the CANCEL command for server termination. If you CANCEL a server address space, the logical sequence of a transaction can be interrupted, leaving the transaction incomplete. If you do CANCEL a server, you must ensure that no records are left in a checked out state. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for more information.

## Creating Additional Copies of a Server

Each transaction request from NetView is queued in the bridge dispatcher until the NetView Bridge Adapter server becomes available. With a high volume of transaction traffic, your users might experience increased response times. During high traffic times, you may want to create more than one copy of an adapter server. Multiple copies provide greater throughput for your transactions by giving the bridge dispatcher more than one server to send the queued requests to. Keep in mind, however, that each copy of a server costs you in terms of storage and possible degradation of performance of both NetView and Tivoli Information Management for z/OS.

To create an additional copy of an existing NetView Bridge Adapter server, make a separate copy of the JCL procedure and modify the copy as follows:

1. Assign a unique receiver PPI queue name for each additional server.
2. Define a unique APPLID for each extra server.
3. Define unique output data sets for each server's use. See "The User-Supplied Output Data Sets" on page 17.

   **Note:** If you want to run a second copy of the adapter with different initialization parameters, you can modify the INFOBRDS data set or create a new copy of the data set and point to it in the modified JCL procedure. If you want to run two copies of the adapter with the same parameters, both can use the same INFOBRDS data set.
4. Add automation logic to start each additional copy of the server. See step 6 in "Setting Up the NetView Bridge Adapter" on page 7.

You do not need to change any programming of NetView commands or NetView Bridge definitions to use the additional copies of a server.

## Recycling the Servers

You must recycle the servers that you create any time that the NetView Subsystem Address Space is recycled. The transactions that the servers are processing, which are stored in the Subsystem Address Space, are lost whenever the address space recycles.

# Performing the NetView Bridge Adapter Transactions

The Tivoli Information Management for z/OS NetView Bridge Adapter provides three transaction processors: Create Record, Update Record, and Retrieve Record by Argument. You can create your own transaction processors to use information in the database. These transactions are requested from NetView command procedures.

Each transaction type has a request form and a reply form. This section contains details that are the same for all three NetView-provided transaction types and user-written transactions. The first topic of this chapter explains these details. The sections following explain each transaction type individually. Finally, the last section contains detailed information about three complex parameters that are used by more than one transaction type.

## Coding the Transactions

Two basic steps are required in a NetView command procedure to use the NetView Bridge Adapter. First, after preparing all the input data needed to make the request, the NetView command procedure starts the NetView Bridge Requester API to generate and transmit the NetView Bridge Adapter transaction request. Then, if the command procedure expects a reply, it must receive the reply before processing its contents.

You have a choice when it comes to writing a NetView command procedure. You can write it as a High-Level Language (HLL) command (in C or PL/I), or you can write it in REXX or NetView Command List Language. Refer to the NetView manuals *NetView Customization: Using PL/I and C* and *NetView Customization: Writing Command Lists* for more information on writing command procedures.

These two versions of NetView command procedures have differences in the way they process adapter transactions. The following descriptions of generating a transaction request and receiving a transaction reply discuss these differences. After this section, all references to the header parameters use the command list form of the name.

### Generating NetView Bridge Adapter Requests

A NetView command procedure written in one of the HLL styles performs transaction requests by calling the NetView Bridge Requester API routine CNMSNDT. (Note that procedures written in C language do not capitalize the entire name of any routine. For example, CNMSNDT is Cnmsndt in the C language procedure.) If the command procedure is written in REXX or NetView Command List Language, it calls the routine TRANSND. You can find information about the format and usage of the parameters in each of these calls in the *NetView Bridge Implementation* manual.

### Header Parameters

You use three types of parameters on a call to generate a transaction request: header parameters, control parameters, and input parameters. Some header parameters are used exclusively by the NetView Bridge and some contain specific information that the NetView Bridge Adapter and the HLAPI use. In a call to the NetView Bridge Requester API procedure CNMSNDT, the last parameter is a pointer to a linked list of parameter blocks. The command list version of this command procedure, which calls TRANSND, makes the last parameter a list of parameters.

### Control Parameters and Input Parameters

For both versions of the command procedure, these parameter lists consist of the two other types of parameters that the NetView Bridge Adapter requires: control parameters and input parameters. The control parameters are used by the NetView Bridge Adapter and the HLAPI to determine how to process the input parameters. The input parameters are Tivoli Information Management for z/OS field names that contain the data that the system uses to process database records. Each transaction requires specific control and input parameters.

Control and input parameters are processed by the NetView Bridge Adapter. Because control parameters are transaction specific, they are described later in this chapter in the description of each of the transaction types. In some cases the adapter translates input parameters into LLAPI Program Interface Data Table (PIDT) names and processes them as described in the *Tivoli Information Management for z/OS Application Program Interface Guide*. You must format input parameters according to the corresponding type of Tivoli Information Management for z/OS field. These formats include response type inputs, list type inputs, and text type inputs. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for more information about these inputs.

You need to understand list inputs because Tivoli Information Management for z/OS processes some of its fields as list data. In all transaction requests you must specify a SEPARATOR control parameter. When formatting list parameters, you define the parameter items with that separator character between them. For example,

```
SEPARATOR = '*'
S1416 = 'DEV1*DEV5*DEV8'
```

shows that you have defined the asterisk (*) as your separator, and listed DEV1, DEV5, and DEV8 as your list data. For information on how to format list contents, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

Text inputs are used for database record fields that contain descriptive text. The NetView Bridge Adapter processes these fields on input and output as array parameters. For a more complete discussion of array parameters, refer to the *NetView Bridge Implementation* manual. On input, you must create a control parameter TEXTLIST whose value is a list of the names of any array-type parameters in the transaction. You separate the names with a blank. You also must provide the values of each element in the array. You can define one correct value for TEXTLIST that all NetView commands accessing the database can use. Include in it all s-word indexes that are defined as text fields along with their aliases. See “TEXTLIST” on page 37 for more information about this parameter.

The following examples show how to prepare text input data. You can use C, PL/I, and REXX, but NetView Command List Language does not support array data-entry.

**REXX**

```
USE_DATA_VIEW = 'YES'          /* Use Data Views */
TEXTLIST='S0E01 S0E02'
S0E01.0 = 2
S0E01.1 = 'First line of S0E01'
S0E01.2 = 'Second line of S0E01'
S0E02.0 = 1
S0E02.1 = 'The only line of S0E02'
PARMVAR = 'USE_DATA_VIEW TEXTLIST S0E01. S0E02.'
```

**PL/I**

```
TEXTLIST='S0E01 S0E02';
alloc dsipprm set (parmlist);
parmlist -> prmname = 'S0E01';
parmlist -> prmindex = 1;
parmlist -> prmptr = addr(parm1);

alloc dsipprm set (parmlist -> prmlink);
parmlist -> parmlist -> prmlink;
parmlist -> prmname = 'S0E01';
parmlist -> prmindex = 2;
parmlist -> prmptr = addr(parm2);

alloc dsipprm set (parmlist -> prmlink);
parmlist -> parmlist -> prmlink;
parmlist -> prmname = 'S0E02';
parmlist -> prmindex = 1;
parmlist -> prmptr = addr(parm3);
parmlist -> prmlink = NULL;
```

**C**

```
strcpy(textlist,S0E01 S0E02);
strcpy(parm1.parmname,S0E01);
parm1.prmindex := 1;
parm1.prmlink := &parm2;
prmptr := "line 1 of S0E01";

strcpy(parm2.parmname,S0E01);
parm2.prmindex := 2;
parm2.prmlink := &parm3;
prmptr := "line 2 of S0E01";

strcpy(parm1.parmname,S0E02);
parm3.prmindex := 1;
parm3.prmlink := NULL;
prmptr := "line 1 of S0E02";
```

Table 1 and Table 2 on page 24 show the parameters that you use to write a NetView command procedure in one of the HLL forms. You use these parameters to call module CNMSNDT (Cnmsndt in C language) of the NetView Bridge API. For information on how to specify these parameters, refer to *NetView Customization: Using PL/I and C*.

*Table 1. HLL Command Processor Header Parameters*

| Parameter Name | Description | Specified for |
|---|---|---|
| hlbptr | Anchor pointer used by the NetView HLL API. | NetView Bridge |
| sttransid | Name of the transaction to be run. | NetView Bridge Adapter |
| sttrtype | Type of transaction reply processing to be performed. | NetView Bridge, NetView Bridge Adapter |

*Table 1. HLL Command Processor Header Parameters (continued)*

| Parameter Name | Description | Specified for |
|---|---|---|
| stvocab | Name of the alias table for this transaction. If you leave this parameter blank, the HLAPI does not perform any alias table processing for this transaction. | HLAPI |
| stcorr | Name of the correlation parameter that enables the requesting command and any replies to match each other. | NetView Bridge |
| stdesttask | Destination task ID. This is the task ID of the NetView autotask that accesses the NetView Bridge Adapter servers. | NetView Bridge |
| stdestnet | Destination network ID. If you do not specify an ID, this parameter defaults to the local network ID. | NetView Bridge |
| stdestdom | Destination domain ID. If you do not specify an ID, this parameter defaults to the local domain ID. | NetView Bridge |
| stparms | Pointer to a linked list of parameter blocks. Format of these blocks is shown in Table 2. | NetView Bridge |

*Table 2. HLL Command Processor Parameter Block Format*

| Field Name | Description | Specified for |
|---|---|---|
| prmlink | A link to the next parameter block in the chain. | NetView Bridge |
| prmindex | The index value of the parameter, if it is an array. The only parameters processed by NetView Bridge Adapter as arrays are those used for text fields. For all non-array fields, the value of the prmindex is —32 767. See “TEXTLIST” on page 37 for more information on arrays. | NetView Bridge, NetView Bridge Adapter |
| prmname | The parameter name. It can be an s-word index, a p-word index, or an alias, if there is an active alias table. The maximum number of characters supported for this field is 31. The adapter cannot send or receive 32-character aliases. | NetView Bridge Adapter |
| prmnaml | The length of the parameter name. | NetView Bridge |
| prmtype | The type of the parameter. All NetView Bridge Adapter data is represented in EBCDIC character strings. This field should always be set to X’*00EE*’ to indicate EBCDIC type. | NetView Bridge, NetView Bridge Adapter |
| prmptr | A pointer to the value of the parameter. | NetView Bridge, NetView Bridge Adapter |
| prmleng | The length of the parameter value. | NetView Bridge |

Table 3 on page 25 shows the parameters that you use to write a NetView command procedure in NetView Command List Language or REXX. You use these parameters to call module TRANSND of the NetView Bridge API.

*Table 3. Format for Header Parameters for Command Procedure Written in NetView Command List Language or REXX*

| Parameter Name | Description | Specified for |
|---|---|---|
| TRANSID | Name of the transaction to be run. | NetView Bridge Adapter |
| TRTYPE | Type of transaction reply processing to be performed. | NetView Bridge, NetView Bridge Adapter |
| VOCAB | Name of the alias table for this transaction. If you omit this optional parameter, the HLAPI does not perform any alias table processing for this transaction. | HLAPI |
| CORR | Name of the correlation parameter that enables the requesting command and any replies to match each other. | NetView Bridge |
| DESTTASK | Destination task ID. This is the task ID of the NetView autotask that accesses the NetView Bridge Adapter servers. | NetView Bridge |
| DESTNET | Destination network ID. If you do not specify an ID, this parameter defaults to the local network ID. | NetView Bridge |
| DESTDOM | Destination domain ID. If you do not specify an ID, this parameter defaults to the local domain ID. | NetView Bridge |
| PARMVAR | A list of names of control and input parameters separated by blanks. The NetView Bridge treats each name as a REXX/NetView Command List Language variable. Refer to the NetView Bridge documentation to see how to specify array and non-array variables. See "TEXTLIST" on page 37 to understand how the NetView Bridge Adapter uses array parameters. | NetView Bridge |

## Receiving NetView Bridge Adapter Replies

This transaction returns the reply results of the operation to the requester. The three types of parameters returned to the requester are header parameters, output parameters, and result parameters. For the command list type of command procedure, use the NetView Bridge Requester API routine TRANRCV to retrieve all header, output, and result parameters. For command procedures written in high-level languages, use the NetView Bridge Requester API CNMGETP. Details about using these routines are located in the *NetView Bridge Implementation* manual.

An additional header parameter, RESPCODE, returns information about the success or failure of the request processing. RESPCODE contains values of return codes; these are explained in "NetView Bridge Adapter Codes" on page 59. Either the HLAPI or the NetView Bridge Adapter generates the value of RESPCODE, depending on the results of processing. Its field value is right justified with the leading characters set to blanks.

The output parameters are specific to each transaction. To see which parameters apply to which transaction type, see:
- "Creating a Record (IBCREATE transaction)" on page 27
- "Updating a Record (IBUPDATE Transaction)" on page 31
- "Retrieving a Record by Argument (IBSEARCH Transaction)" on page 34
- "Writing User-Defined Transactions" on page 44.

The result parameters are informational parameters that can return from any transaction type. Table 4 shows the possible result parameters. When the parameter RESPCODE contains a failure code, the REASONCODE parameter leads you to information that explains the failure of the transaction. If processing detects a possible problem, but can continue to perform, it can return a WARNING result parameter. Warnings may show up even if processing is successful and both RESPCODE and REASONCODE have success codes in them. Input parameters in a NetView Bridge Adapter request might have errors. The return parameters BADPARM and ERRPARM contain information that indicates which request parameters are not correct.

When you use the NetView Command List Language or REXX, you must use the NetView Bridge Requester API command TRANRCV to generate return data. This command stores the returned data into variables that are generated at run-time. Refer to the *NetView Bridge Implementation* manual for information on processing the returned data. If you use an HLL command procedure to obtain transaction reply data, the procedure differs from procedures written in NetView Command List Language or REXX. When using the HLLs, the command procedure obtains the values of the header parameters by calling the NetView Bridge Requester API service routine CNMGETP (Cnmgetp in C language) using its option H. To obtain result and output parameters, use options N or P of the routine CNMGETP. Refer to the *NetView Bridge Implementation* manual for more information.

Table 4 describes possible result parameters from all transactions.

*Table 4. Result Parameters from Transactions*

| Parameter Name | Description | Length |
|---|---|---|
| MESSAGES | If the value of the field hli_msg in INFOBRDS is not *P*, then Tivoli Information Management for z/OS returns messages. Otherwise, the messages are discarded. This parameter is an array parameter. Each message that returns is an entry in the array. The array element with an index value of zero returns the number of elements in the array. | 256 or less |
| REASONCODE | This parameter contains a value explaining the success or failure of the transaction. For specific values of reason codes, see NetView Bridge Adapter Codes. | 8 |
| WARNING | This parameter contains a value that specifies a possible problem situation. For specific values of warning codes see Table 19 on page 62. | 8 |
| ZEROLENGTH | This parameter contains a value identifying the name of a parameter that a NetView Bridge Adapter request specifies. The value of the specified parameter has a length of zero. The NetView Bridge Adapter rejects this request. | 31 or less |

*Table 4. Result Parameters from Transactions (continued)*

| Parameter Name | Description | Length |
|---|---|---|
| BADPARM | This parameter contains a value that identifies a parameter that a NetView Bridge Adapter request specifies. When the HLAPI detects an error condition for that parameter as it is anchored to the HICA input chain field, it puts the identity of the parameter into BADPARM. The value returns in the format *htid: c-parmname*, where<br>■ *htid* identifies the HLAPI transaction specified when the error is found<br>■ *c-* identifies the parameter data block code (PDBCODE) of the parameter in error<br>■ *parmname* identifies the parameter in error.<br><br>The BADPARM parameter is an array parameter; one or more transaction parameters that are found to be in error by the HLAPI can be entries in the array. The array element with index value of zero returns the number of elements in the array. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for more information. | 40 or less |
| ERRPARM | This parameter contains a value identifying a PIDT error that the HLAPI detects as the result of a transaction. The value returns in the format *htid: pc-pidtentry*, where<br>■ *htid* identifies the HLAPI transaction specified when the error is found<br>■ *pc-* identifies the PIDT code (PIDTCODE) of the PIDT entry in error<br>■ *pidtentry* identifies the PIDT entry in error.<br><br>The ERRPARM parameter is an array parameter; one or more PIDT entries that are found to be in error by the HLAPI can be entries in the array. The array element with index value of zero returns the number of elements in the array. | 41 or less |
| VERIFIER | Name of first verifier parameter that fails verification during IBUPDATE transaction. | 31 or less |

## Creating a Record (IBCREATE transaction)

The Tivoli Information Management for z/OS NetView Bridge Adapter provides the Create Record transaction so that you can make new records in the Tivoli Information Management for z/OS database from the NetView product. Descriptions of the two forms of this transaction, request and reply, follow.

### The IBCREATE Transaction Request

The request transaction creates a Tivoli Information Management for z/OS data record. You can verify that a similar record does not already exist in the database. Because the NetView Bridge Adapter does not support Tivoli Information Management for z/OS's Add Record Relation transaction (HL12), you cannot define relationships between records, such as parent/child relationships using the IBCREATE transaction.

To create a record, the NetView command procedure calls the appropriate NetView Bridge Requester API routine. Which routine is called depends on whether you write the command procedure in the HLL, REXX, or NetView Command List Language style. The routine receives header parameters, control parameters, and input parameters. The header and control parameters contain the information that determines how the transaction creates the record. Tivoli Information Management for z/OS treats any other parameters supplied by this transaction as input and attempts to store them in the created record.

For information on the header parameters that the transaction passes in its call to the NetView Bridge API routines, see "Header Parameters" on page 22. In addition, you need the following header parameter information specifically for a Create Record transaction.

**TRANSID**
The value of this field is IBCREATE.

**TRTYPE**
This parameter indicates the reply processing that the transaction performs. The values for IBCREATE are:

**QERR**
Return only error response data to the requesting task.

**QALL**
Return all operational response data to the requesting task.

**QNON**
Return no responses of either type to the requesting task.

**VOCAB**
The name of VOCAB is an 8-byte field that specifies the name of the alias table used for this transaction. If this parameter is all blanks (or omitted in a REXX or Command List Language call), the Tivoli Information Management for z/OS HLAPI does not perform any alias table processing.

**Note:** INFOBRDS parameter alias_table_cnt is required if you want to use alias table processing.

Table 5 on page 29 contains the control parameters used to issue an IBCREATE request. The NetView Bridge Adapter converts some of the parameters in this table into Tivoli Information Management for z/OS HLAPI control items. For more information, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*. The Tivoli Information Management for z/OS HLAPI creates the record in the Tivoli Information Management for z/OS database. If you request a reply and the create is successful, the record ID of the created record and a return code specifying a successful creation return to the caller.

To perform the optional verification, you must have the NetView command procedure pass a list of search arguments to the routine along with the record information. The adapter initiates a record search. If the search finds one or more records to match all of the search arguments, you receive a list of INQRESULTs listing the records and a reason code that specifies that the record you want to create is not unique. If the search finds no match, the API creates the record in the database and issues a return code indicating a successful creation.

Table 5. Control Parameters for Create Record

| Parameter Name | Description | Length |
|---|---|---|
| INQPRIV | The NetView Bridge Adapter translates this parameter into the Tivoli Information Management for z/OS HLAPI control name PRIVILEGE_CLASS, which is used in an HLAPI transaction that performs an inquiry on a Tivoli Information Management for z/OS database. This parameter is not necessary if performing an unconditional create. If you omit this parameter, the HLAPI transaction uses the initialization privilege (initclas in INFOBRDS) class. | 8 |
| CREPRIV | The NetView Bridge Adapter translates this parameter into the Tivoli Information Management for z/OS HLAPI control name PRIVILEGE_CLASS, which is used in an HLAPI transaction that creates a record in the Tivoli Information Management for z/OS database. If you omit this parameter, the HLAPI transaction uses the initialization privilege (initclas in INFOBRDS) class. | 8 |
| INQVIEW | The NetView Bridge Adapter translates this parameter into the Tivoli Information Management for z/OS HLAPI control name PIDT_NAME, which is used in an HLAPI transaction that performs an inquiry on a Tivoli Information Management for z/OS database. The name specified by INQVIEW defines the record type for the inquiry. This parameter is required only when verifying uniqueness. If BYPASS_PANEL_PROCESSING=YES in INFOBRDS, then the INQVIEW value will be used as a DATA_VIEW_NAME. If BYPASS_PANEL_PROCESSING=NO is used in INFOBRDS, then INQVIEW will be treated as a DATA_VIEW_NAME if the INFOBRDS keyword USE_DATA_VIEW value is YES or an input data item USE_DATA_VIEW is used that has the value of YES. | 8 |
| CREVIEW | This is a required field. The NetView Bridge Adapter translates this parameter into the Tivoli Information Management for z/OS HLAPI control name PIDT_NAME, which is used in a HLAPI transaction that creates a record in the Tivoli Information Management for z/OS database. The name specified by CREVIEW defines the record type for the record created. If BYPASS_PANEL_PROCESSING=YES is specified in INFOBRDS, then the CREVIEW value will be used as a DATA_VIEW_NAME. If BYPASS_PANEL_PROCESSING=NO is specified in INFOBRDS, then CREVIEW will be treated as a DATA_VIEW_NAME if the INFOBRDS keyword USE_DATA_VIEW is YES or an input data item USE_DATA_VIEW is used that has the value of YES. | 8 |

Table 5. Control Parameters for Create Record (continued)

| Parameter Name | Description | Length |
|---|---|---|
| ASSOCDATA | The NetView Bridge Adapter translates this parameter into the Tivoli Information Management for z/OS HLAPI control name ASSOCIATED_DATA, which is used in an HLAPI transaction that performs an inquiry on a Tivoli Information Management for z/OS database. This parameter identifies the name of a field in a matched database record whose contents are returned to the requester as a part of the output parameter INQRESULT. | 31 |
| TEXTLIST | This parameter identifies the input parameter names that correspond to Tivoli Information Management for z/OS text type fields. A more in-depth description of this parameter appears in "TEXTLIST" on page 37. This parameter is required only when creating a record with text field data. | variable |
| SEPARATOR | The NetView Bridge Adapter translates this parameter into the Tivoli Information Management for z/OS HLAPI control name SEPARATOR_CHARACTER, which is used in an HLAPI transaction that performs an inquiry on a Tivoli Information Management for z/OS database and in an HLAPI transaction that creates a record. This parameter identifies the character field value that separates data items in a list. | 1 |

## The IBCREATE Transaction Reply

This transaction returns the reply results of the creation operation to the requester. This reply can return result parameters, which are explained in "Receiving NetView Bridge Adapter Replies" on page 25, as well as those specific output parameters shown in Table 6.

Table 6. Output Parameters for Create Record

| Parameter Name | Description | Length |
|---|---|---|
| RECORDID | Produces a return only if a record is created. This parameter contains the record ID of the created record. | 8 |
| INQRESULT | Produces a return only if a Create Unique Record search finds matching records in the database. The value of this parameter is the same as the INQUIRY_RESULT parameter of the Tivoli Information Management for z/OS HLAPI. This parameter is treated as an array where each element of the array is a separate INQRESULT that matched the search conditions. Thus, three matches produce three instances of the parameter INQRESULT, each with a different parameter index value. An additional INQRESULT instance with a parameter index value of zero returns the number of elements in the array. For more information, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*. | 87 |

## Updating a Record (IBUPDATE Transaction)

The Tivoli Information Management for z/OS NetView Bridge Adapter provides the Update Record Transaction so you can change data in an existing Tivoli Information Management for z/OS record.

### The IBUPDATE Transaction Request

This request transaction updates a Tivoli Information Management for z/OS data record. You can verify that the record to be updated contains the data you expect to find.

To update a record, the NetView command procedure calls the appropriate NetView Bridge Requester API routine. Which routine is called depends on whether you write the command procedure in the HLL, REXX, or NetView Command List Language style. The routine receives header parameters, control parameters, and input parameters. The header and control parameters contain the information that determines how the transaction updates the record. To update a list of items in the record, you must include all items in the list. If no error occurs during processing, Tivoli Information Management for z/OS stores all input parameters in the record and issues a return code indicating successful update.

To perform the optional verification, the transaction sends an additional parameter that contains the names of the fields to be verified, and their expected contents, to the routine. The Tivoli Information Management for z/OS HLAPI compares the contents of these fields to the contents of the record to ensure that they match. If the verification fails, the update does not occur, and the system sets the return and reason codes to indicate failure. If the verification is successful and the update is not, then the return code tells you the update was not successful and the reason code tells you why the update failed. If verification and update are both successful, the transaction updates the record and issues a return code indicating success.

For information on the complete set of header parameters that the transaction passes in its call to the NetView Bridge API routines, see “Generating NetView Bridge Adapter Requests” on page 21. In addition, an Update Record transaction needs the following specific header parameter information:

**TRANSID**
The value of this field is IBUPDATE.

**TRTYPE**
This parameter indicates the transaction reply processing that is performed. The values for IBUPDATE are:

**QERR**
Return only error response data to the requesting task.

**QALL**
Return all operational response data to the requesting task.

**QNON**
Return no responses of either type to the requesting task.

**VOCAB**
The name of VOCAB is an 8-byte field that specifies the name of the alias table used for this transaction. If this parameter is all blanks (or omitted in a REXX or Command List Language call), the Tivoli Information Management for z/OS HLAPI does not perform any alias table processing.

**Note:** INFOBRDS parameter alias_table_cnt is required if you want to use alias table processing.

Table 7 contains the control parameters you use to issue an IBUPDATE request. The NetView Bridge Adapter translates some of the parameters in this table into Tivoli Information Management for z/OS HLAPI control items. For more information, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

*Table 7. Control Parameters for Update Record*

| Parameter Name | Description | Length |
|---|---|---|
| RECORDID | This is a required parameter. This parameter is the identifier for the specific Tivoli Information Management for z/OS database record that is updated. The NetView Bridge Adapter translates this parameter into the Tivoli Information Management for z/OS HLAPI control name RNID_SYMBOL, which is used in HLAPI transactions that check out, retrieve, and update a record in the database. | 8 |
| UPDPRIV | The NetView Bridge Adapter translates this parameter into the Tivoli Information Management for z/OS HLAPI control name PRIVILEGE_CLASS, which is used in HLAPI transactions that check out, retrieve, and update a record in the database. If you omit this parameter, the HLAPI transaction uses the initialization privilege (initclas in INFOBRDS) class. | 8 |
| RETRVIEW | The NetView Bridge Adapter translates this parameter into the Tivoli Information Management for z/OS HLAPI control name PIDT_NAME, which is used in an HLAPI transaction to retrieve a record in the database for verification. The name specified by RETRVIEW defines the record type for the retrieval. This parameter is required only when you perform an update with verification. If BYPASS_PANEL_PROCESSING=YES in INFOBRDS, then the RETRVIEW value will be used as a DATA_VIEW_NAME. If BYPASS_PANEL_PROCESSING=NO is used in INFOBRDS, then RETRVIEW will be treated as a DATA_VIEW_NAME if the INFOBRDS keyword USE_DATA_VIEW value is YES or an input data item USE_DATA_VIEW is used that has the value of YES. | 8 |

*Table 7. Control Parameters for Update Record (continued)*

| Parameter Name | Description | Length |
|---|---|---|
| UPDVIEW | This is a required parameter. The NetView Bridge Adapter translates this parameter into the Tivoli Information Management for z/OS HLAPI control name PIDT_NAME. It is used in an HLAPI transaction that updates a record in the database. The name specified by UPDVIEW defines the record type for the update. If BYPASS_PANEL_PROCESSING=YES in INFOBRDS, then the RETRVIEW value will be used as a DATA_VIEW_NAME. If BYPASS_PANEL_PROCESSING=NO is used in INFOBRDS, then RETRVIEW will be treated as a DATA_VIEW_NAME if the INFOBRDS keyword USE_DATA_VIEW value is YES or an input data item USE_DATA_VIEW is used that has the value of YES. | 7 |
| TEXTLIST | This parameter identifies the input parameter names that correspond to Tivoli Information Management for z/OS text type fields. A more in-depth description of this field appears in "TEXTLIST" on page 37. This field is required only when updating text fields. | variable |
| REPLACE_TEXT_DATA | This is an optional parameter. The NetView Bridge Adapter adds this parameter to the HLAPI as a control name. A value of YES will cause existing text data in the record to be replaced. Any other value is ignored and text data will be appended to existing data. | 3 |
| SEPARATOR | This is a required parameter. The NetView Bridge Adapter translates this parameter into the Tivoli Information Management for z/OS HLAPI control name SEPARATOR_CHARACTER, which is used in the HLAPI transaction that updates a record in a database. This parameter identifies the character field value that separates data items in a list.<br><br>A single separator character in this field indicates that the response item in the record is to be deleted. See Figure 2 on page 34 for an example.<br>**Note:** The INFOBRDS parameter VALIDATE must be set to NO if you want to delete response items. | 1 |
| VERIFIER | This parameter contains arguments needed to verify that a Tivoli Information Management for z/OS database record contains the expected data. More discussion of this field appears in "VERIFIER" on page 43. | variable |

Figure 2 shows three update transactions updating an existing list of routine names. For each transaction, the figure shows: the list before the transaction on the left, the response buffer segment used to update the list, and the results of the update.

```
List Before    Response Buffer       List After    Action Per-
Update         Segment               Update        formed


ADD            ',,,'                 --------      Deleted first 3
BUILD1                               --------      items on list
DELITEM                              --------
COPY                                 COPY
--------                             --------
CHECK                                CHECK
INIT                                 INIT



ADD            'ADD,BUILD1,,COPY'    ADD           Deleted third
BUILD1                               BUILD1        item on list
DELITEM                              --------
COPY                                 COPY
--------                             --------
CHECK                                CHECK
INIT                                 INIT



ADD            ' , , , , , ,'        ADD           Deleted seventh
BUILD1                               BUILD1        item on list
DELITEM                              DELITEM
COPY                                 COPY
--------                             --------
CHECK                                CHECK
INIT                                 --------
```

*Figure 2. List Item Update Example*

A single separator character as a response for a nonlist item indicates that this response item in the record is to be deleted.

### The IBUPDATE Transaction Reply

This transaction returns the reply results of the update operation. “Receiving NetView Bridge Adapter Replies” on page 25 tells about result parameters that can be returned with this reply. For the Update Record transaction, no transaction-specific output parameters exist.

## Retrieving a Record by Argument (IBSEARCH Transaction)

The Tivoli Information Management for z/OS NetView Bridge Adapter provides the Retrieve Record by Argument transaction so you can search a Tivoli Information Management for z/OS database for one or more specific records.

### The IBSEARCH Transaction Request

This request transaction retrieves a Tivoli Information Management for z/OS data record by search arguments. If you give a specific RECORDID as a search argument, then the transaction retrieves that specific record and no search occurs. In this instance, you must specify the RECORDID in the SEARCHLIST as RECORDID=*rnid*, where RNID can be mixed data. In all other cases the transaction performs a search using the search arguments you specify with SEARCHLIST. The transaction uses the input parameters you specify to identify the chosen record fields that the search returns to you.

If the search finds a single match, the system returns the RECORDID and the requested contents of the matching record. Only those fields you specify as input parameters have their data returned to you. But, if you do not identify specific fields in your request, the transaction returns the contents of the record as specified by the parameter RETRVIEW. If the search finds more than one match, the transaction returns an array of INQRESULTs. If no matches occur, the transaction returns a failure return code.

**Note:** You cannot request the specific retrieval of text type fields. Any field that you request that is not found, or is of type text, returns in the parameter BADPARM.

For information on the header parameters that the transaction passes in its call to the NetView Bridge Requester API routines, see “Header Parameters” on page 22. In addition, the Retrieve Record transaction needs the following specific header parameter information:

**TRANSID** The value of this field is IBSEARCH.

**TRTYPE** This parameter indicates the transaction processing that is performed. The only valid value for this transaction is QALL. QALL specifies that the transaction return all operational response data to the requesting task.

**VOCAB** The name of VOCAB is an 8-byte field that specifies the name of the alias table that is used for this transaction. If this parameter is all blanks (or omitted in a REXX or Command List Language call), the HLAPI does not perform any alias table processing.

**Note:** INFOBRDS parameter alias_table_cnt is required if you use alias table processing.

Table 8 contains the control parameters used to issue an IBSEARCH request. The HLAPI uses any other parameters supplied by this transaction to identify the selected parameters that are returned to the requester. The NetView Bridge Adapter translates some of the parameters in this table into HLAPI control items. For more information, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

*Table 8. Control Parameters for Retrieve Record by Argument*

| Parameter Name | Description | Length |
|---|---|---|
| INQPRIV | The NetView Bridge Adapter translates this parameter into the HLAPI control name PRIVILEGE_CLASS, which is used in an HLAPI transaction that performs an inquiry on a database. This parameter is not necessary if you are performing a retrieval of a specific record by RECORDID. If you omit this parameter, the HLAPI transaction uses the initialization privilege (init_class in INFOBRDS) class. | 8 |
| RETRPRIV | The NetView Bridge Adapter translates this parameter into the HLAPI control name PRIVILEGE_CLASS, which is used in an HLAPI transaction that retrieves a record in the database. If you omit this parameter, the HLAPI transaction uses the initialization privilege (initclas in INFOBRDS) class. | 8 |

*Table 8. Control Parameters for Retrieve Record by Argument (continued)*

| Parameter Name | Description | Length |
|---|---|---|
| INQVIEW | The NetView Bridge Adapter translates this parameter into the HLAPI control name PIDT_NAME, which is used in an HLAPI transaction that performs an inquiry on a database. The name that INQVIEW specifies defines the record type for the inquiry. This parameter is not necessary if you are performing a retrieval of a specific record by RECORDID. If BYPASS_PANEL_PROCESSING=YES in INFOBRDS, then the INQVIEW value will be used as a DATA_VIEW_NAME. If BYPASS_PANEL_PROCESSING=NO is used in INFOBRDS, then INQVIEW will be treated as a DATA_VIEW_NAME if the INFOBRDS keyword USE_DATA_VIEW value is YES or an input data item USE_DATA_VIEW is used that has the value of YES. | 8 |
| RETRVIEW | This is a required parameter. The NetView Bridge Adapter translates this parameter into the HLAPI control name PIDT_NAME, which is used in an HLAPI transaction that retrieves a record in the database. The name that RETRVIEW specifies defines the record type for the retrieval. If BYPASS_PANEL_PROCESSING=YES in INFOBRDS, then the RETRVIEW value will be used as a DATA_VIEW_NAME. If BYPASS_PANEL_PROCESSING=NO is used in INFOBRDS, then RETRVIEW will be treated as a DATA_VIEW_NAME if the INFOBRDS keyword USE_DATA_VIEW value is YES or an input data item USE_DATA_VIEW is used that has the value of YES. | 8 |
| ASSOCDATA | The NetView Bridge Adapter translates this parameter into the HLAPI control name ASSOCIATED_DATA, which is used in an HLAPI transaction that performs an inquiry on a database. This parameter identifies the name of a field in the database record whose contents are returned to the requester along with the record identifier as a part of the output field INQRESULT. | 31 |
| SEPARATOR | This is a required parameter. The NetView Bridge Adapter translates this parameter into the HLAPI control name SEPARATOR_CHARACTER, which is used in the HLAPI transaction that performs an inquiry on a database. This parameter identifies the character field value that is used to separate data items in a list. | 1 |
| SEARCHLIST | This is a required parameter. This string contains search arguments that identify a specific database record. A more in-depth description of this parameter appears in "SEARCHLIST" on page 38. | variable |

## The IBSEARCH Transaction Reply

This reply transaction returns the results of the retrieve transaction. "Receiving NetView Bridge Adapter Replies" on page 25 explains the result parameters that can be returned with this reply. Also, this transaction can return those specific output parameters shown in Table 9.

Table 9 shows the output parameters that are specific to the transaction Retrieve Record by Argument.

*Table 9. Output Parameters for Retrieve Record by Argument*

| Parameter Name | Description | Length |
|---|---|---|
| INQRESULT | Provides results only if the transaction does not specify a RECORDID in the SEARCHLIST parameter and the search finds matching records. The value for this parameter matches the value in the HLAPI parameter INQUIRY_RESULT. The NetView Bridge Adapter treats this parameter as an array where each element of the array is a separate INQRESULT that matches the search conditions. Thus, three matches produce three instances of the parameter INQRESULT, each with a different parameter index value. An additional INQRESULT instance with a parameter index value of zero returns the number of elements in the array. For more information, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*. | 87 |
| SEPARATOR | The NetView Bridge Adapter translates this parameter from the output HLAPI parameter name SEPARATOR_CHARACTER, which identifies the character the transaction uses to separate each of the data responses in a list of items. | 1 |

## Using Complex Parameters

Each of the following complex parameters appears in more than one of the transactions that the Tivoli Information Management for z/OS NetView Bridge Adapter supplies. To see which parameter applies to which transaction, see:

- "Creating a Record (IBCREATE transaction)" on page 27
- "Updating a Record (IBUPDATE Transaction)" on page 31
- "Retrieving a Record by Argument (IBSEARCH Transaction)" on page 34.

### TEXTLIST

The parameter TEXTLIST identifies to the NetView Bridge Adapter transaction processors which parameter names are text fields in the Tivoli Information Management for z/OS database records. Unlike other fields that contain a relatively small (fewer than 120) number of characters, text data fields consist of large numbers of lines of variable format containing from 0 to 132 characters per line. The transaction passes each line of text to the NetView Bridge Adapter separately as an element of an array with a parameter name that is found in the TEXTLIST parameter. If you write programs in the NetView Command List Language or REXX that call a transaction, the program must set the zero element of the array to the number of lines of text sent to NetView. The Adapter gathers these lines together and passes them to the HLAPI, which stores them with the rest of the data in the Tivoli Information

Management for z/OS record. When the program receives lines of freeform text from NetView, NetView places the freeform text into an array with a parameter name that is found in the TEXTLIST parameter. NetView places each line of freeform text into a separate element of the array. The value in the zero element of the array is the number of lines of freeform text that NetView sends to the program. For more information, refer to the TRANSND and TRANRCV commands in the *NetView Bridge Implementation* manual.

The NetView Bridge Adapter cannot access either the alias table information or the information that identifies s-word or p-word indexes that correspond to text data items. Therefore, you must identify those fields that contain text data so the NetView Bridge Adapter can handle them properly. The TEXTLIST parameter allows you to do this.

TEXTLIST is a list of parameter names; each one represents a text data field. Each name is followed by one or more blanks to separate it from its successor. It does not include array indexing information because the NetView Bridge Adapter processes *all* elements of an array that is sent as part of a transaction. This list can contain more parameter names than the transaction requested uses, so a fixed TEXTLIST parameter can be used by many transactions.

Parameter names for text fields are identified by taking the parameter name in the transaction and finding that name in the TEXTLIST parameter before any alias processing, so the parameter names passed in the transaction must be identical to those in TEXTLIST. If the parameter you use in the transaction is an alias name, then you must use the alias name in the TEXTLIST. If you use an s-word index in the transaction, then the TEXTLIST name must be the s-word index. In other words, if you want to fill a text field with your transaction, the text field name must appear in TEXTLIST.

## SEARCHLIST

A SEARCHLIST consists of a series of search words; each word defines a search condition. You enter each search word in a SEARCHLIST as a series of characters with no imbedded blanks. These search words are similar to the specification of Tivoli Information Management for z/OS search arguments using the terminal interface. The *Tivoli Information Management for z/OS User's Guide* describes three methods of searching (structured, freeform, and combined) and three types of search words (structured, prefixed, and freeform). The NetView Bridge Adapter uses search types similar to those in the Tivoli Information Management for z/OS, and the HLAPI adds the ability to create alias s-word and p-word indexes.

The following sections of this chapter tell you about the three search methods you can perform and give examples of how to use the various search words involved with each method.

### Structured Search Requests

For a structured search request, you use structured search words. The structure's format is: `keyword=value`.

**keyword**
This part of the search word identifies which field in the Tivoli Information Management for z/OS database to compare to the value field in this search word. The four ways to define *keyword* in this case are s-word index, s-word index alias, p-word index, and p-word index alias.

**s-word index**
Use the s-word index that Tivoli Information Management for z/OS PIDT defines for this field.

**p-word index**
Use the p-word index that Tivoli Information Management for z/OS PIDT defines for this field.

**s-word/p-word index**
Use the s-word or p-word index that Tivoli Information Management for z/OS PIDT defines for this field.

**alias** Use the alias keywords to identify the field to be searched. The alias table identified with this transaction defines these words.

**=** The equal sign means to process this search word as a structured inquiry.

**value** This field specifies the value to use in the comparison for each record to determine whether an inquiry returns that record. This value can be:
Mixed data
SBCS data (excluding SO/SI characters)
DBCS data

The abbreviated keyword character (**.**) and the position ignore character (*****) are not treated as special characters when used in *value* as part of a structured search word.

Some examples of valid SBCS structured search words are:

- S0B59=JONES

  This is an s-word that retrieves all records that have the value *JONES* in the field that the s-word index S0B59 identifies.

- ABC=def

  Assuming that ABC is an alias keyword of S0B59, this is an s-word that retrieves all records that have the value *def* in the field that the s-word index S0B59 identifies.

For a structured search response, the words are not processed in the sequence in which they appear in the SEARCHLIST, but in the order in which they appear in the Tivoli Information Management for z/OS PIDT that you identify with the INQVIEW parameter. Because structured searches use only implied logical ANDs, the order of processing is important only because it may affect the sequence in which multiple results return.

## Freeform Search Requests

You can do a freeform search request with two types of search words: prefixed and freeform.

### Prefixed Search Word

The format for a prefixed search word is `+#keyword/value`.

**+** When the SBCS plus character (+) appears in the first position of a search word, the remainder of the word is used as the search word. You must use this character when you do not specify *keyword* as a prefix alias.

**#** The search operator field. This is an optional field. If you omit this field, the SEARCHLIST treats the condition this word specifies as a logical AND with the other search conditions. This field is one of the following:

**|** Use the SBCS vertical bar when this search condition is a logical OR in context with the other conditions of this SEARCHLIST. You cannot use this operator on the first freeform word in a SEARCHLIST.

Use the SBCS logical not symbol when a record with this field is to be included in the inquiry results list, but only when the specified value does not match the value of the field in the record.

**-** Use the SBCS hyphen when the value of this field is the end of a range for the previous reference to this field. Do not use this in the first freeform word in a SEARCHLIST.

**Note:** SEARCHLIST handles the absence of this search operator (**#**) by treating the condition this word specifies as a logical AND with other search words.

**keyword**
This part of the search word identifies which field in the Tivoli Information Management for z/OS database to compare to the value field in this search word. It must be an SBCS string. Keyword may be defined as either a p-word or an alias defined for either a p-word or a prefix index.

**p-word** Use the Tivoli Information Management for z/OS-defined p-word for this field. This field is the only place in the NetView Bridge Adapter that supports prefix keywords. You can use a *position ignore* character (*****) in the prefix. The plus character (**+**) should always begin a prefix search word. For information on how prefix keywords can affect a search, refer to the *Tivoli Information Management for z/OS User's Guide*.

**prefix alias** Use the alias name that you define in the alias table specified by VOCAB to be equivalent to a prefix keyword or prefix index for this field.

**/** Use the SBCS slash symbol when you want to process this search word as a freeform request. It is optional for prefixed search words if you omit the value field.

**value** This field specifies the value to use in the comparison for each record to determine if an inquiry returns that record. This value can be:
Mixed data
SBCS data (excluding SO/SI characters)
DBCS data

The abbreviated keyword character (**.**) and the position ignore character (*****) are supported in this method of searching.

Some examples of valid SBCS prefixed search words are:

- ABC/def

  If ABC is a prefix alias of P01AC, which in turn is an index to PERS/, then the search word is converted to PERS/def, and it retrieves all records that contain PERS/def.

- PERSO/JONES

  If PERSO is an alias of the prefix PERS/, this is a freeform word that retrieves all records that contain PERS/JONES. If PERSO is not an alias or alias prefix, this argument is not valid and causes an error.

- +PERS/JONES

  Assuming that PERS/ is a prefix, this is a prefixed search word that returns any records that contain PERS/JONES.

- PERSON_OPEN_PROBLEM/JONES

  Assuming that PERSON_OPEN_PROBLEM is an alias prefix of P01AC, which in turn is an index to PERS/, this prefixed search word returns the same records as +PERS/JONES.

- +PER*/JONES

  This prefixed search word returns any record with the value JONES that has a 4-character prefix field starting with PER, such as PERA/, PERS/, PERR/, but *not* PER/.

- +MIS

  This prefixed search word returns any record that has a prefix field starting with MIS, such as MISB/JONES, MISX/JONES, and MISCELLANEOUS_PROBLEM/SMITH.

- +PERS/*ONES

  This is a prefixed search word that finds records with data 5 characters long ending in ONES, with a prefix of PERS/. Examples are PERS/BONES, PERS/CONES, and PERS/JONES.

- +PH/444-.

  This is a prefixed search word that finds records with a prefix of PH/ that start with 444-. Examples are PH/444-4158, PH/444-7126ALTERNATE, and PH/444-6.

- +¬PERS/JONES

  This is a prefixed search word that returns any records that do not contain the prefix field PERS/JONES.

- +RNID/00000000 +-RNID/00000100

  This is a prefixed search word that retrieves all records whose record IDs fall between 0000 and 0100.

- +PERS/JONES +|PERS/SMITH

  This is a prefixed search word that retrieves all records that have either JONES or SMITH values in PERS.

## Freeform Search Word

The second type of search word that you can use in a freeform search request is the freeform search word. The format for a freeform search word is `#/value`

`#` The search operator field. This is an optional field. If you omit this field, the SEARCHLIST treats the condition this word specifies as a logical AND with the other search conditions. This field is one of the following:

> `|` Use the SBCS vertical bar when this search condition is a logical OR in context with the other conditions of this SEARCHLIST. You cannot use this operator on the first freeform word in a SEARCHLIST.
>
> Use the SBCS logical not symbol when a record with this field is to be included in the inquiry results list, but only when the specified value does not match the value of the field in the record.

**Note:** The SEARCHLIST handles the absence of this search operator (**#**) by treating the condition this word specifies as a logical AND with other search words.

**no keyword entry**
A freeform search is done using the string you specify in the value part of the search word to check description fields and abstract fields for the data. Thus, no keyword is needed. This is equivalent to using freeform words in an interactive search.

**/** Use the SBCS slash symbol when you want to process this search word as a freeform request. It is optional for freeform search words.

**value** This field specifies the value to use in the comparison for each record to determine if an inquiry returns that record. This value can be:
Mixed data
SBCS data (excluding SO/SI characters)
DBCS data

The abbreviated keyword character (**.**) and the position ignore character (*****) are supported in this method of searching.

Some examples of valid SBCS freeform search words are:

- ¬abc

  This is a freeform search word that retrieves all records that do not have abc in any non-prefixed field. For example, text fields are non-prefixed fields.
- abc

  This is a freeform search word that retrieves all records that have abc in any non-prefixed field.
- /abc

  This is a freeform search word that retrieves all records that have abc in any non-prefixed field.
- /pink ¬blue

  This freeform search word retrieves the records that have pink but do not have blue in any non-prefixed field.
- pink ¬/blue

  This freeform search word retrieves the records that have pink but do not have blue in any non-prefixed field.
- blue |/p*nk

  This freeform search word retrieves the records that have blue or 4-character data beginning with *p* and ending with *nk* in any non-prefixed field. This search returns, for example, pink, punk, and ponk.

For a freeform search response, the words are processed in the sequence that they appear in the SEARCHLIST (from left to right). This may be significant to the results of a search because the results depend upon the sequence of application of logical ANDs, ORs, and NOTs.

**Note:** Because the search does not permit parentheses, this sequence must be made carefully to achieve the results you expect.

### Combined Structured and Freeform Search Requests

You may combine structured and freeform search words in a single SEARCHLIST.

An example of a valid combined search word is:

- S0B59=JONES +RNID/00000050 -00000100

  These are combined structured and freeform search words that find records with the field S0B59 having a value of JONES within the record number range of 50 to 100.

For a combined search response, all s-words in the SEARCHLIST are processed first. The remaining words in the SEARCHLIST are appended to the s-word list and the HLAPI performs the search.

The HLAPI is the primary performer of SEARCHLIST processing. The NetView Bridge Adapter takes the input string value of SEARCHLIST and converts it into the format expected as input by the HLAPI. The HLAPI translates the parameter names based upon the specified VOCAB parameter before performing the Tivoli Information Management for z/OS database inquiry.

The NetView Bridge parameter VOCAB and its counterpart the ALIAS_TABLE PDB in the HLAPI identify an alias table that the HLAPI uses in all searches for records that it retrieves from the database. Each database input parameter that the HLAPI processes is put through this table. If the input parameter does not match an entry in the PIDT or alias table, an error returns to the requester.

## VERIFIER

The VERIFIER parameter consists of a series of verifier words separated by blanks. Each verifier word is a character string that defines a verification condition. A successful verification requires that each word that the VERIFIER parameter specifies must be satisfied. A string can contain blanks if the value is delimited by single quotation mark or double quotation mark characters. The format of the verifier word is `parmname=value`.

**parmname**
This required field identifies the Tivoli Information Management for z/OS database field that the HLAPI compares to the value part of this condition. This field must be defined consistently with the alias table referenced with this transaction. You can perform verification against fields of data type *blank*, data type *date*, or data type *list* only. For the list data type, the search examines each item in the list. If it finds a match, the search ends with a successful verification. Other types of fields, such as string, phrase, text, and direct add fields, fail verification.

**=**
The equal sign is required in the format of this word.

**value**
This field specifies the value to use in the comparison to determine if a record meets the verification conditions. This value can be:

Mixed data
SBCS data (excluding SO/SI characters)
DBCS data

The abbreviated keyword (.) and the position ignore (*) characters receive no special consideration during verification. They appear as any other character.

If you want to verify that a field is null, mark the value portion of the verifier word with empty quotation marks. For example,

```
VERIFIER='S0C3D=""'
```

or

```
VERIFIER="S0C3D=''"
```

verifies that the record has no value in field S0C3D.

If this field begins with a single quotation mark or a double quotation mark character, the next appearance of the quotation mark must be at the end of the field and a blank must follow it, or it must be the last item in the verifier word. You cannot imbed quotation mark characters within the value field if they are the same type of quotation mark characters that you use to start the field. So, in the examples above the *empty* quotation marks are the opposite of those used to delimit the contents of the VERIFIER word itself.

## Writing User-Defined Transactions

The Tivoli Information Management for z/OS NetView Bridge Adapter supports three transactions to create, update and retrieve records in the Tivoli Information Management for z/OS database. Sometimes you may want to perform other operations on this data. For example, you may want to define relationships between records, such as parent/child relationships. The Tivoli Information Management for z/OS NetView Bridge Adapter provides support to enable you to define and use your own NetView Bridge Adapter transactions. A *user-defined transaction* is a set of tasks that the user defines to be performed on records in a centralized Tivoli Information Management for z/OS database. User-defined transactions can use all API functions in the HLAPI *except* the HLAPI transactions Initialize Tivoli Information Management for z/OS (HL01) and Terminate Tivoli Information Management for z/OS (HL02).

A *user-supplied module* is a user-written routine that implements one or more user-defined transactions. When the NetView Bridge Adapter reads a transaction identifier that it does not recognize and you have specified a user-supplied module to NetView, the NetView Bridge Adapter calls that module to process the transaction.

The user-supplied module calls three NetView Bridge Adapter routines to access the NetView Bridge Adapter interface services. These routines and the tasks they perform are:

| | |
|---|---|
| **BLGBURC** | Retrieves NetView transaction data through the NetView Bridge Adapter and builds a chain of PDBs for use by the HLAPI |
| **BLGBUIM** | Calls the Tivoli Information Management for z/OS HLAPI for processing user-defined transactions contained in the user-supplied module |
| **BLGBUSN** | Sends response data through the NetView Bridge Adapter to NetView using a chain of PDBs. |

These routines are packaged together in one module, BLGBUSR, which is link-edited in the target library when you install the NetView Bridge Adapter. See “Installing the NetView Bridge Adapter” on page 7 for information on installing the NetView Bridge Adapter.

The user-supplied module uses Tivoli Information Management for z/OS HLAPI Parameter Data Blocks (PDBs), not NetView *Message Parameter Blocks* (MPBs). The NetView Bridge

Adapter routine BLGBURC converts MPBs into PDBs, and the NetView Bridge Adapter routine BLGBUSN converts PDBs into MPBs to move information between NetView and Tivoli Information Management for z/OS.

## A Typical Scenario

A typical scenario in using this support follows.

1. An application programmer writes a re-entrant C-language routine (a user-supplied module) that receives information from the NetView application and returns information to the NetView application. The module includes calls to the NetView Bridge Adapter interface services and defines codes to be returned to the NetView Bridge Adapter at exit.
2. In NetView, the programmer writes an application to enable NetView to send information to the user-supplied module through the NetView Bridge Adapter. The application interprets the information returned by the user-supplied module.
3. In Tivoli Information Management for z/OS, the programmer compiles the user-supplied module and link-edits it with BLGBUSR into the target library. The programmer also adds the module name to the INFOBRDS transaction_processor parameter. To improve performance, the programmer sets the INFOBRDS parameter preload_tran_proc parameter to YES.
4. When the NetView Bridge Adapter is initialized, the user-supplied module is loaded into storage because the preload_tran_proc parameter has been set to YES.
5. At the appropriate time, NetView sends the transaction request to the Tivoli Information Management for z/OS NetView Bridge Adapter.
6. The NetView Bridge Adapter processes the transaction request through the Tivoli Information Management for z/OS HLAPI.
7. The HLAPI returns a response to the NetView Bridge Adapter, which processes the information according to the user-supplied module and sends it back to NetView.

## Requirements for a User-Supplied Module

The following sections describe the requirements for writing a user-supplied module and the two forms of the transactions that it can support, request and reply.

The NetView Bridge Adapter can call a user-supplied module that has the following characteristics:

- Can process one or more transactions.
- Written in PL/I, C, or assembler language.
- Reside in a load library accessible to the MVS LINK macro.
- Link-edited with the BLGBUSR module.
- Before ending, this module must release all storage it obtains, and any storage obtained at its request such as the PDB chain received from BLGBURC.

If the user-supplied module is re-entrant it is not refreshed between calls, and any changes made to it are not in effect until the NetView Bridge Adapter is stopped and restarted.

## Identifying the User-Supplied Module

The INFOBRDS data set has two parameters that refer to the user-supplied module:

|  |  |
|---|---|
| **transaction_processor** | Indicates to the NetView Bridge Adapter that a user-supplied module exists. You must specify the name of the user-supplied module on this parameter. |
| **preload_tran_proc** | Indicates when to load the user-supplied module into storage. |

**YES** The user-supplied module is loaded into storage (but not called) when the NetView Bridge Adapter is initialized. The module must be re-entrant because it remains in storage until the NetView Bridge Adapter is stopped.

**NO** The user-supplied module is loaded into storage and deleted each time it is called. This method can result in significant system overhead, but the module does not have to be re-entrant. The default value for preload_tran_proc is NO.

Refer to the "The User-Supplied Input Data Set" on page 10 for more information on specifying these parameters.

The user-supplied module is called only when both of the following conditions are met:

- The name of the module is specified in the INFOBRDS transaction_processor parameter.
- The NetView Bridge Adapter receives a transaction identifier that is not IBCREATE, IBSEARCH or IBUPDATE.

If the NetView Bridge Adapter reads an unknown transaction identifier and a module name has not been specified in the INFOBRDS data set, the NetView Bridge Adapter returns the return code and reason code for an unknown transaction identifier.

## Writing User-Supplied Modules

User-supplied modules pass data between NetView and Tivoli Information Management for z/OS using a Transaction Processor Communication Area control block (TPCA) that contains the fields listed below. The user-supplied module can change any field in the TPCA that is in the following list.

*Table 10. Transaction Processor Communications Area (TPCA)*

| Field Label | Offset DEC(HEX) | Length DEC | Type | Description | Set by |
|---|---|---|---|---|---|
| BLGBTPCA | 0(0) | 128 | STRUCTURE | Transaction Processor Communication Area | -- |
| TPCAID | 0(0) | 4 | CHARACTER | Control block identifier | User-supplied module |
| TPCALEN | 4(4) | 4 | SIGNED | Control block length | User-supplied module |
| TPCATRID | 8(8) | 8 | CHARACTER | Transaction identifier from NetView | Caller |
| TPCARETC | 16(10) | 4 | SIGNED | Transaction return code | NetView Bridge Adapter interface services |
| TPCAREAC | 20(14) | 4 | SIGNED | Transaction reason code | NetView Bridge Adapter interface services |
| TPCAHLRC | 24(18) | 4 | SIGNED | Tivoli Information Management for z/OS HLAPI return code | Tivoli Information Management for z/OS HLAPI |

Table 10. Transaction Processor Communications Area (TPCA) (continued)

| Field Label | Offset DEC(HEX) | Length DEC | Type | Description | Set by |
|---|---|---|---|---|---|
| TPCAHLRS | 28(1C) | 4 | SIGNED | Tivoli Information Management for z/OS HLAPI reason code | Tivoli Information Management for z/OS HLAPI |
| TPCACTLP | 32(20) | 4 | ADDRESS | Address of the first control PDB | User-supplied module |
| TPCAINP | 36(24) | 4 | ADDRESS | Address of the first input PDB | User-supplied module |
| TCPAOUTP | 40(28) | 4 | ADDRESS | First results PDB | BLGBUIM |
| TCPAMSGP | 44(2C) | 4 | ADDRESS | First message PDB | BLGBUIM |
| TPCAERRP | 48(30) | 4 | ADDRESS | First error PDB | BLGBUIM |
| TPCAAPPL | 52(34) | 8 | CHARACTER | Application ID | User-supplied module, NetView Bridge Adapter |
| TPCAPRCL | 60(3C) | 8 | CHARACTER | Privilege class from INFOBRDS | NetView Bridge Adapter |
| TPCAOPT | 68(44) | 8 | CHARACTER | IBRPRINT option from INFOBRDS | NetView Bridge Adapter |
| TPCAJOBN | 76(4C) | 8 | CHARACTER | Bridge server job name | NetView Bridge Adapter |
| TPCARESP | 84(54) | 8 | CHARACTER | Transaction response code | User-supplied module |
| TPCAIUO | 92(5C) | 36 | CHARACTER | Reserved | -- |

The NetView Bridge Adapter interface services builds a parameter list that contains the address of the TPCA. The address of the parameter list is passed to the user-supplied module in register 1.

The user-supplied module must perform the following steps:

1. Call the BLGBURC routine to retrieve information sent by the NetView application.
2. Build two PDB chains: one for control PDBs and one for input PDBs for calling the HLAPI transactions.
3. Call the BLGBUIM routine to perform the appropriate HLAPI transactions. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for information on using HLAPI transactions.
4. Build one PDB chain from the PDB chains returned from BLGBUIM in the previous step and from any PDBs that contain information that the user-supplied module might need to return to the caller.
5. If the module sends several PDBs with the same name to NetView, the module must set the PDBAPPL field in each of these PDBs to a unique index value.
6. Call the BLGBUSN routine to send a reply to the NetView application.
7. Release all storage this module obtains and all storage obtained for this module, including the PDB chain received from the BLGBURC module.

Information on performing these steps follows.

## Calling the BLGBURC Routine

The BLGBURC routine retrieves request data through the NetView Bridge Adapter and builds a chain of PDBs for use by the HLAPI. It also sets a pointer variable to locate the first PDB in the chain.

The user-supplied module specifies two parameters when it calls the BLGBURC routine. For example, the following code segment calls this routine:

```
BLGBURC(TPCA_ADDRESS, PDB_ADDRESS)
```

where `TPCA_address` is the address of the TPCA passed to the user-supplied module by the NetView Bridge Adapter interface services, and `PDB_address` identifies the first PDB in a chain of PDBs created by the BLGBURC routine. The BLGBURC routine sets the `PDB_address` parameter for use later in the user-supplied module; the module does not initialize it.

## Building Control and Input PDB Chains

The PDB chain created by the BLGBURC routine cannot be processed directly by the HLAPI. After calling the BLGBURC routine, code in the user-supplied module must use the PDBs returned by the BLGBURC to build separate control and input PDB chains for use by the BLGBUIM routine. The contents of the PDBs in each PDB chain depends on the request parameters sent from the NetView application.

**Note:** The user-supplied module must identify the type of each PDB returned from the BLGBURC routine to correctly build the PDB chains for the BLGBUIM routine. These PDBs are either control or input PDBs.

After the user-supplied module builds the two PDB chains, it places the address of the first control PDB in the TPCA field TPCACTLP, and places the address of the first input PDB in the TPCA field TPCAINP.

## Calling the BLGBUIM Routine

The BLGBUIM routine calls the HLAPI for processing information in the Tivoli Information Management for z/OS database. The user-supplied module specifies one parameter when calling the BLGBUIM routine: the address of a TPCA. The TCPA contains:

- The address of the first PDB of a control PDB chain
- The address of the first PDB of an input PDB chain.

For example, the following code segment calls this routine:

```
BLGBUIM(TPCA_ADDRESS)
```

where `TPCA_address` is the address of the TPCA passed to the user-supplied module by the NetView Bridge Adapter interface services.

Upon return from BLGBUIM, the TPCA contains the following:

- A return code and reason code set by the HLAPI
- Addresses of the first PDBs in three PDB chains: the message, error, and output PDB chains.

## Building a Results PDB Chain

The user-supplied module receives three PDB chains from the BLGBUIM routine. The user-supplied module uses these PDB chains to build one PDB chain for use by the BLGBUSN routine. The user-supplied module can add PDBs to this results PDB chain in addition to selecting any PDBs returned from the BLGBUIM routine.

**Note:** The user-supplied module must ensure that the PDB chains returned from the BLGBUIM routine remain intact. The next HLAPI transaction to run frees storage for these PDBs, but if they have been modified, unpredictable results can occur.

### Setting the PDBAPPL Field

PDBs with the same name contain the same value in the PDBNAME field. These PDBs must contain a unique value in the PDBAPPL field. The user-supplied module sets consecutive values in this field starting at zero.

### Calling the BLGBUSN Routine

The BLGBUSN routine sends response data through the NetView Bridge Adapter to NetView. Before calling this routine, the user-supplied module must set a value in the TPCARESP field in the TPCA. The user-supplied module specifies two parameters when it calls the BLGBUSN routine. For example, the following code segment calls this routine:

```
BLGBUSN(TPCA_address, PDB_address)
```

where `TPCA_address` is the address of the TPCA passed to the module by the NetView Bridge Adapter interface services and `PDB_address` identifies the first PDB in a valid, intact PDB chain containing PDBs returned from the BLGBUIM routine. It can also contain PDBs built by the user-supplied module.

## NetView Return Codes

The user-supplied module receives return codes from the NetView Bridge Adapter in register 15 and in the TPCA. Both must be examined by the module to determine the status of processing. In addition, the module receives reason codes from the BLGBUIM routine in the TPCA. Descriptions of these return codes and reason codes follow.

### Return Codes in Register 15

When the NetView Bridge Adapter returns control to the user-supplied module, it passes a return code in register 15. These return codes and their meaning are:

| | |
|---|---|
| **0** | The transaction was successful. Continue with the next transaction. |
| **12** | The address of the TPCA is less than 4096. |
| **16** | The first 4 characters in the TPCA are not **TPCA**. |

### Return and Reason Codes in TPCA

When the user-supplied module calls a NetView Bridge Adapter service, that service passes a return code and a reason code to the user-supplied module in the TPCA. These return codes and reason codes are described below.

- Return codes from BLGBURC:

  | | |
  |---|---|
  | **0** | The service was performed as requested. |
  | **8** | CNMEGTP completed with an unexpected return code indicating a possible NetView error. The TPCA contains a NetView reason code. |

- Return codes from BLGBUIM:

  | | |
  |---|---|
  | **0** | The service was performed as requested. |
  | **4** | The service was performed as requested, but the HLAPI issued a nonzero return code. |
  | **8** | The requested HLAPI transaction is not supported. |

- Reason codes from BLGBUIM:

  | | |
  |---|---|
  | **0** | The service was performed as requested. |
  | **4** | The HLAPI transaction HL01 is not supported. |
  | **8** | The HLAPI transaction HL02 is not supported. |

- Return codes from BLGBUSN:

**0** The service was performed as requested.
**8** CNMESTR completed with an unexpected return code indicating a possible NetView error. The TPCA contains a NetView reason code.

### User-Supplied Module Return Codes

When all processing in the module is completed, you must pass a return code to the NetView Bridge Adapter in the TPCA that indicates the success or failure of processing. The allowable values for this return code and their meaning are:

**0** The transaction was successful. Continue with the next transaction.
**4** The transaction code is unknown. Continue with the next transaction.
**8** The transaction failed. Stop further NetView Bridge Adapter processing.

## Sample Code for the User-Supplied Module

The following section describes sample panel and code parts that demonstrate how to create and run a user-supplied module. The Tivoli Information Management for z/OS SAMPLIB (MVS data set SBLMSAMP) contains these parts. You can print these parts just as you would any file on MVS. The parts are listed and described in Table 11.

*Table 11. Sample parts in SAMPLIB*

| Part Name | | | Description |
|---|---|---|---|
| BLGNBREX | | | A REXX EXEC that starts the user-supplied module. It runs under NetView. |
| BLGCCHK | BLGCCRT | BLGCRET | NetView panels that pass information to the user-supplied module. |
| BLGCINQ | BLGCOBT | BLGCTRAN | |
| BLGCTSP | BLGCUPD | BLGHFCRT | |
| BLGHFDEL | BLGHFFFT | BLGHFINQ | |
| BLGHFLDA | BLGHFLDB | BLGHFLD1 | |
| BLGHFLD2 | BLGHFOBT | BLGHFRET | |
| BLGHFTSP | BLGHFUPD | BLGHMAIN | |
| BLGHPCRT | BLGHPINQ | BLGHPMSG | |
| BLGHPORT | BLGHPPF1 | BLGHPPF2 | |
| BLGHPREC | BLGHPRET | BLGHPSTA | |
| BLGHPUPD | BLGICRT | BLGIFFT | |
| BLGIINQ | BLGIRET | BLGISTAT | |
| BLGIUPD | BLGOFFT | BLGOMSG | |
| BLGORECS | BLGORET | | |
| BLGNBSRC | | | Source code for the user-supplied module. |
| BLGCTPCA | BLGCPDB | | C language macros to define the TPCA and PDB structures. |
| BLGNBLNK | | | JCL to link-edit the user-supplied module with the BLGBUSR module. |

After you start the NetView Bridge Adapter, perform the following steps to use the user-supplied module:

1. Place the REXX EXEC BLGNBREX in a command list library that is listed in the DSICLD DD statement in your NetView procedure CNMPROC.

2. Place the NetView panels in a library that is listed in the CNMPNL1 DD statement in your NetView procedure CNMPROC.
3. Modify the BLGNBREX REXX EXEC as follows:
   a. Find DESTTASK.
   b. Change BRIDGE1 to the name of the NetView Bridge that is defined in the DSIPARM NetView data set.
   c. Save your changes.
4. Modify the BLGNBLNK JCL as follows:
   a. Provide valid information on the JOB card.
   b. Reference the appropriate data sets on your system.
   c. Provide on the SYSLMOD DD statement the name of a data set listed in the STEPLIB DD statement of the started task JCL for the Tivoli Information Management for z/OS NetView Bridge Adapter.
5. Modify the INFOBRDS data set to include the following lines:
   ```
   TRANSACTION_PROCESSOR='NVBACSMP'
   PRELOAD_TRAN_PROC='NO'
   ```
6. Stop and restart the NetView Bridge Adapter.
7. Logon to NetView.
8. Type **BLGNBREX** on the NetView command line and press Enter to start the user-supplied module.

## Calling User-Defined NetView Bridge Adapter Transactions

The Tivoli Information Management for z/OS NetView Bridge Adapter provides support for calling your own transactions. Descriptions of the two forms of calling user-defined transactions, request and reply, follow.

### The User-Defined Transaction Request

The request transaction manipulates the Tivoli Information Management for z/OS database as specified in the transaction module.

To perform the work, the NetView command procedure calls the appropriate user-supplied module. Which module is called depends on whether you write the command procedure in the HLL, REXX, or NetView Command List Language style. The module receives header parameters, input parameters, and user-defined control parameters. The header parameters contain the information that determines how the information is processed. You must communicate with the transaction programmer to find out what other parameters must be supplied for this transaction, including user-defined control parameters.

The following header parameter information is specifically required for a user-defined transaction.

**TRANSID** The value of this field is the name of the user-defined transaction.

**TRTYPE** This parameter indicates the reply processing that the transaction performs. The values for user-defined transactions are:

**QERR**
Return only error response data to the requesting task.

**QALL**
Return all operational response data to the requesting task.

**QNON**
Return no responses of either type to the requesting task.

**VOCAB** An 8-byte field that contains a user-defined value that is used for this transaction. If this parameter is all blanks (or omitted in a REXX or Command List Language call), the Tivoli Information Management for z/OS HLAPI does not perform any alias table processing.

**Note:** INFOBRDS parameter alias_table_cnt is required if you want to use alias table processing.

Most of the header parameters that are required for a user-defined transaction are also required by the IBCREATE, IBUPDATE, and IBSEARCH transactions. The VOCAB parameter can contain any value for a user-defined transaction. The NetView Bridge Adapter considers all other parameters on a user-defined transaction to be input parameters. They are passed as elements in a PDB chain via the PDB input and output services between NetView and the user-defined transaction. The NetView Bridge Adapter does not examine input parameters. The user-defined transaction can specify which input parameters are used as control parameters.

## The User-Defined Transaction Reply

This reply transaction returns the results of the user-defined transaction to the requester. The results are returned in the results PDB chain. The user-supplied module builds the results PDB chain from the error and message PDB chains returned to it, and any PDBs built by the user-supplied module, to pass other information back to the requester.

4

# The NetView Bridge Adapter IBRPRINT Data Set

## Contents of the IBRPRINT Data Set

The data set with the DDNAME of IBRPRINT stores the information that you specify about NetView Bridge Adapter transactions.

You determine what is stored in this data set at the initialization of the NetView Bridge Adapter by setting the variable IBRPRINT_OPTION in the data set INFOBRDS. If you set the variable to ALL, then the system uses IBRPRINT to record transaction requests, replies, and errors. If you set the variable to ERROR, then IBRPRINT will show only error data. If you set the IBRPRINT_OPTION to NONE, then the system records no transaction data.

Each record in IBRPRINT has a 32-byte header containing control information followed by 72 bytes of input data. The header format is the same for each record, and it consists of the type of record (request, reply, error, or data set error), the date and time, and the application identifier (APPLID). The format of the 72 bytes of input information depends on the type of record. The following tables show the header format and the format for each of the record types.

*Table 12. IBRPRINT Record Header Format*

| Field Name | Description | Length | Value |
|---|---|---|---|
| RECTYPE | Type of record. Either request (REQST), reply (REPLY), error (ERROR), or data set error (DSERR). | 5 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| DATE | Date record was made. Character string of year, month, day. | 6 | yymmdd |
| BLANK | Single blank to separate fields. | 1 | |
| TIME | Time record was made. Character string of hours, minutes, seconds, and milliseconds. | 9 | hhmmssttt |
| BLANK | Single blank to separate fields. | 1 | |
| APPLID | Application identifier. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |

Table 13 shows the format of the 72 bytes of input data in an IBRPRINT transaction request record.

Table 13. IBRPRINT Transaction Request Record Format

| Field Name | Description | Length | Value |
|---|---|---|---|
| TRANSID | Transaction identifier of the request. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| ORIGDOM | Originating domain of the request. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| ORIGTASK | Originating task of the request. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| ORIGNET | Originating network identifier of the request. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| CORRELATOR | Character string that relates request to reply. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| TRANSTYPE | Transaction type of the request. Character string. | 4 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| ALIAS | Alias table name from the request. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| GPCURSOR | The returned value of DSINBEG’s *gpcursor*. | 8 | Input |
| FILLER | Blank filler area to maintain log size of 104 bytes. | 5 | |

Table 14 shows the format of the 72 bytes of input data in an IBRPRINT transaction reply record.

Table 14. IBRPRINT Transaction Reply Record Format

| Field Name | Description | Length | Value |
|---|---|---|---|
| TRANSID | Transaction identifier of the reply. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| DESTDOM | Destination domain of the reply. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| DESTASK | Destination task of the reply. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| DESTNET | Destination network identifier of the reply. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| CORRELATOR | Character string that relates reply to request. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |

*Table 14. IBRPRINT Transaction Reply Record Format (continued)*

| Field Name | Description | Length | Value |
|---|---|---|---|
| RESPONSE | Response code for the reply. Blank if the IBRPRINT REASON field contains a warning code. | 4 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| REASON | Reason code or warning code for the reply. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| RESERVED | Reserved. | 8 | Reserved |
| FILLER | Blank filler area to maintain log size of 104 bytes. | 5 | |

Table 15 shows the format of the 72 bytes of input data in an IBRPRINT transaction error record.

*Table 15. IBRPRINT Transaction Error Record Format*

| Field Name | Description | Length | Value |
|---|---|---|---|
| TRANSID | Transaction identifier of the transaction request being processed when the error occurs. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| ORIGDOM | Originating domain of the transaction request. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| ORIGTASK | Originating task of the transaction request. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| ORIGNET | Originating network identifier of the transaction request. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| CORRELATOR | Correlator from the transaction request. Character string. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| FAILEDID | Character string that identifies the Adapter module that returned an error or caused the Adapter to detect an error. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| ERRCODE | Numeric error code. | 4 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| REASON | Numeric reason code. | 8 | Reserved |
| FILLER | Blank filler area to maintain log size of 104 bytes. | 5 | |

Table 16 shows the format of the 72 bytes of input data in an IBRPRINT transaction data set allocation error record.

Table 16. IBRPRINT Transaction Data Set Allocation Error Record Format

| Field Name | Description | Length | Value |
|---|---|---|---|
| DSNAME | Character string that gives the DSNAME of the data set where the error occurred. | 44 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| RESERVED | Reserved. | 4 | Reserved |
| BLANK | Single blank to separate fields. | 1 | |
| FAILEDID | Character string that identifies the Adapter module that returned an error or caused the Adapter to detect an error. | 8 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| ERRCODE | Numeric error code. | 4 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| REASON | Numeric reason code. | 8 | Input |

These are examples of request, error, and reply records that can appear in the IBRPRINT data set. Each record is 104 bytes long and appears as a single line on the terminal screen when you look at it.

```
'REQST 901231 120340619 TEMP     IBSEARCH CNM01    OPERATOR NETA
  18120340 QALL ALIAS003 35674814     '

'ERROR 901231 120341011 TEMP     IBSEARCH CNM01    OPERATOR NETA
  18120340 HLL        12       40     '

'REPLY 901231 120341023 TEMP     IBSEARCH CNM01    BRIGOPER NETA
  18120340   12       40              '
```

Table 17 shows the format of the 72 bytes of input data in an IBRPRINT transaction trace record.

Table 17. IBRPRINT Transaction Data Set Trace Record Format

| Field Name | Description | Length | Value |
|---|---|---|---|
| TRACED PARAMETER | Name of the function being traced. **Note:** Currently only the input parameters are traced. | 13 | Fixed value. Example: ‘INPUT-CNMGETP’ |
| BLANK | Single blank to separate fields. | 1 | |
| TRACER IDENTIFIER | Name of the module tracing when recording a parameter value. | 8 | Fixed character name of CSECT |
| BLANK | Single blank to separate fields. | 1 | |
| PARAMETER ID | Identifier of the parameter being recorded. | 31 | Input |
| BLANK | Single blank to separate fields. | 1 | |
| PARAMETER VALUE | Value of the parameter data being recorded. **Note:** Only 45 characters of data are displayed because of the way the IBRPRINT data set is recorded. | 45 (max) | Input |
| BLANK | Single blank to separate fields. | 1 | |
| REASON | Numeric reason code. | 8 | Input |

These are examples of trace entries that you can use in the IBRPRINT data set.

```
INPUT-CNMGETP      BLGBEXT      CRVIEW       BLGYPR
INPUT-CNMGETP      BLGBEXT      TEXTLIST     S0E01
INPUT-CNMGETP      BLGBEXT      SEPARATOR    ,
INPUT-CNMGETP      BLGBEXT      S0B59        BROWNJ
INPUT-CNMGETP      BLGBEXT      S0BEE        INITIAL
INPUT-CNMGETP      BLGBEXT      S0E0F        This is a test
INPUT-CNMGETP      BLGBEXT      S0E01        First line of text
```

5

# NetView Bridge Adapter Codes

## Return Codes and Reason Codes

Table 18 defines the return codes and reason codes that the NetView Bridge Adapter sends to the NetView requester (if it requests a reply) and records in the log (if you have set one up). These codes can be generated by the Adapter, NetView, and the Tivoli Information Management for z/OS HLAPI. For more information on NetView and Tivoli Information Management for z/OS HLAPI codes, refer to the *NetView Bridge Implementation* and *Tivoli Information Management for z/OS Application Program Interface Guide*.

*Table 18. NetView Bridge Adapter Return Codes and Reason Codes*

| Return Code (RESPCODE) | Reason Code | Description |
|---|---|---|
| 0 | 00 | Successful completion. |
| 2 | all codes | Transaction completed normally, but results were unsuccessful. |
| | 20 | More than one record matched the search criteria. |
| | 21 | Record not found in the database using search arguments specified in the SEARCHLIST control parameter. |
| | 22 | Verification failed. Database field value does not match verifier field value. |
| | 23 | One or more records found matching the create unique search criteria. |
| | 24 | Verification word name not found in RETRVIEW table or in the alias table. For more information on verification, see "VERIFIER" on page 43. |
| 4 | all codes | Warning was detected in Tivoli Information Management for z/OS HLAPI. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*. |
| 8 | all codes | Validation error was detected in Tivoli Information Management for z/OS HLAPI. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*. |

*Table 18. NetView Bridge Adapter Return Codes and Reason Codes (continued)*

| Return Code (RESPCODE) | Reason Code | Description |
|---|---|---|
| 10 | all codes | Error was detected in NetView Bridge Adapter. Stop the transaction. |
| | 30 | Required control parameter CREVIEW is missing. |
| | 31 | Required control parameter INQVIEW is missing. |
| | 32 | Required control parameter RETRVIEW is missing. |
| | 33 | Error was detected in the search list. |
| | 34 | Required control parameter SEARCHLIST is missing. |
| | 35 | Required control parameter SEPARATOR is missing. |
| | 36 | TRTYPE is not set to QALL for retrieve. |
| | 37 | TRANSID specified by requester is not known. |
| | 38 | Required control parameter UPDVIEW is missing. |
| | 39 | Text data length greater than 132 characters. |
| | 40 | A transaction parameter was received with no parameter name. |
| | 41 | Input data parameter has a length field equal to zero. |
| | 42 | Binary data was received from NetView Bridge. |
| | 43 | Nonzero processing code was returned in INQUIRY_RESULT data block. |
| | 44 | Required control parameter RNID_SYMBOL is missing. |
| | 45 | Verification was not performed because verifier word name is a nonverifiable field. |
| | 46 | No data in string given for verification. |
| | 47 | Verifier word in verification string not valid, lacks = sign. |
| | 48 | Incorrect mixed data in parameter data extracted from NetView Server Support API. |
| 11 | all codes | Error was detected in NetView Bridge. Stop the transaction. Refer to the *NetView Bridge Implementation* manual. |
| 12 | all codes | Error was detected in Tivoli Information Management for z/OS HLAPI or LLAPI. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*. |
| 16 | all codes | Error was detected in Tivoli Information Management for z/OS HLAPI. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*. |

Table 18. NetView Bridge Adapter Return Codes and Reason Codes (continued)

| Return Code (RESPCODE) | Reason Code | Description |
|---|---|---|
| 18 | all codes | Severe NetView Bridge Adapter error was detected. Close the address space. |
| | 60 | Allocation of the working data set APPLID.TEXTnnnn for the TEXTLIST parameter failed. |
| | 61 | Data set allocation failed. |
| | 62 | Reserved for system use. |
| | 63 | Extract Macro failed on ECB address fetch. |
| | 64 | Error occurred in freeing an adapter PDB. |
| | 65 | Error occurred in freeing an MPB. |
| | 66 | Parameter list on call to HLAPI not valid. |
| | 67 | Error in opening the data set with DDNAME INFOBRDS. |
| | 68 | Error in reading the data set with DDNAME INFOBRDS. |
| | 69 | Required INFOBRDS parameter INITCLAS is missing. |
| | 70 | JCL EXEC parameter has jobname that is not valid. |
| | 71 | Load of Tivoli Information Management for z/OS translate table failed. |
| | 72 | Required INFOBRDS parameter SEND_QUEUE is missing. |
| | 73 | Required INFOBRDS parameter SESSMEMB is missing. |
| | 74 | Register 15 error occurred in call to NetView. |
| | 75 | Reserved for system use. |
| | 76 | Reserved for system use. |
| | 77 | Trying to create a data set, but all data sets are in use. More than 9 text working data sets have been allocated. |
| | 78 | NetView Bridge Adapter has shut down all processing in an orderly manner. |
| | 79 | Could not link to the Tivoli Information Management for z/OS HLAPI. |
| 18 | 80 | The session member specified in the NVBA INFOBRDS data set could not be loaded. Make sure that the session member is in a data set in the STEPLIB concatenation for the NVBA. |
| | 81 | The date exit specified in the session member cannot be loaded. Make sure that the date exit exists in one of the libraries specified in the STEPLIB concatenation for the NVBA. |
| | ≥1000 | Subtract 1000. Look up resulting value in the *NetView Application Programming Guide: Program-to-Program Interface* |
| 20 | all codes | Severe NetView Bridge error was detected. Close the address space. Refer to the *NetView Bridge Implementation* manual. |

## Warning Codes

Table 19 shows the warning codes that the NetView Bridge Adapter generates, returns to the requester, and records in the IBRPRINT data set. Like reason codes, these codes return as result parameters in the transaction reply.

*Table 19. NetView Bridge Adapter Warning Codes*

| Warning Code | Description |
|---|---|
| 15 | Superfluous search criteria was specified with RECORDID. |
| 25 | Some requested parameters could not be retrieved. Check BADPARMs. |
| 71 | Maximum number of inquiry results was exceeded. Available memory exhausted. |

## Log Only Codes

Table 20 shows the log only codes that the NetView Bridge Adapter generates and records in the log data set. Unlike the other codes, these codes are only recorded in the log; they do not return to the requester.

*Table 20. NetView Bridge Adapter Log Only Codes*

| Recorded Code | Description |
|---|---|
| 01 | Error occurred opening the data set with DDNAME IBRPRINT. |
| 02 | Error occurred opening DSNAME. |
| 03 | Error occurred in reading DSNAME. |
| 04 | Error occurred in writing DSNAME. |
| 05 | Error occurred in unallocating DSNAME. |

# II — NetView AutoBridge

6

# NetView AutoBridge Overview

Tivoli Information Management for z/OS NetView AutoBridge is a set of routines, panels, and tables that serve as an application enabler for the Tivoli Information Management for z/OS NetView Bridge Adapter. AutoBridge receives data from specific alerts, messages, and other applications via its application programming interface (API) and then uses this data to build and perform Tivoli Information Management for z/OS transactions.

AutoBridge is an open, modular, expandable, and flexible application interface. AutoBridge builds transactions from the data it receives according to instructions you supply in its process and mapping tables. AutoBridge's functionality, data input, and record output can be customized by system integrators and users.

**Note:** The Tivoli Information Management for z/OS NetView AutoBridge should not be confused with the NetView Bridge Adapter. The NetView Bridge Adapter provides a connection between the NetView Bridge and the Tivoli Information Management for z/OS High-Level Application Program Interface (HLAPI) to access the database; the NetView Bridge Adapter has been a component of Tivoli Information Management for z/OS since Information/Management Version 4.2.2.

The Tivoli Information Management for z/OS NetView AutoBridge provides a different function, as described in the following section. The Tivoli Information Management for z/OS NetView AutoBridge was previously released as a standalone product, and was incorporated into Tivoli Information Management for z/OS only as of Information/Management Version 6 Release 2 Modification 1 (6.2.1).

## What Does Tivoli Information Management for z/OS NetView AutoBridge Do?

Tivoli Information Management for z/OS NetView AutoBridge lets you use the Tivoli Information Management for z/OS NetView Bridge Adapter to automate database operations. With AutoBridge you can automatically create, update, or search database records using data from the following:

- Messages accessible to NetView (VTAM, JES)
- Management services units (MSUs)
- Data from other applications.

AutoBridge uses instructions that you specify to generate a transaction record from captured messages, MSUs, or application data. Depending on the data, AutoBridge will create, search, or update records in a Tivoli Information Management for z/OS database on a local or remote NetView. AutoBridge saves a record of each transaction, so that those that are unsuccessful can be retried. AutoBridge can also post-process transactions to supplement or modify the transaction data as would occur if it were entered by an operator using the Tivoli Information Management for z/OS entry panels.

## Tivoli Information Management for z/OS NetView AutoBridge Highlights

AutoBridge exploits the features of the NetView Bridge in the following ways:

- It creates, updates, and searches records in a Tivoli Information Management for z/OS database.
- It can access database servers that reside on remote hosts.
- It allows access to multiple database servers.
- It returns messages to the originating user or task.

AutoBridge provides the following functions:

- It dispatches records based on priority, resource type, and other user specifications.
- A process table lets you define how different types of input data are processed by specifying a step-by-step list of functions to be performed such as PARSE, TRACE, and CREATE. It can also process user command functions and NetView commands.

  The process table allows input data to be uniquely identified; for example, alerts, messages, and application data can be mapped according to their source of origin.
- A mapping table lets you specify how input data is mapped to alias names, identify search and update fields, and specify additional commands to be run. Segments of this mapping table can include definitions for:
  - Management Services Units (MSUs)
  - Alert messages (BNJ146I)
  - Other messages
  - Application-generated data, such as Automated Operations Network/MVS (AON/MVS) control file information and vital product data
- A filter table allows filtering of input records based on field values that you specify.
- AutoBridge provides a panel interface for loading/replacing, verifying, and displaying AutoBridge's tables.
- AutoBridge's checkpoint-manager automatically copies records to a file for retransmission when a transaction fails.
- AutoBridge's PostProcessor facility simulates operator entry of AutoBridge-created records to allow terminal simulator panel (TSP) and exit program invocation.

## Why Use Tivoli Information Management for z/OS NetView AutoBridge?

AutoBridge can help automate network management tasks such as:
- Monitoring the network for specific events
- Creating and updating Tivoli Information Management for z/OS records
- Searching for duplicate records
- Notifying vendors of the status of their products.

### Managing Network Events

*Network management* can be described as the process of identifying and responding to events that occur on the network. An *event*, used in this context, is the change in status of any element of the network. Because this book uses the term *network* to include a digital communications network and all systems connected to it, network events can include such things as:

- A loss of connection
- A hardware failure
- A modem time-out
- A printer running out of paper.

Network events are displayed to NetView operators as alerts, messages, or data from other applications. Without automation, network operators must manually enter records of these events in the Tivoli Information Management for z/OS database.

AutoBridge automates network monitoring and record creation, so that network information can be entered quickly and accurately into Tivoli Information Management for z/OS. This automation can improve tracking of network events and reduce the amount of monitoring network operators must perform.

### Interfacing with Tivoli Information Management for z/OS

AutoBridge uses the Tivoli Information Management for z/OS NetView Bridge Adapter to send requests to Tivoli Information Management for z/OS. Requests go from the NetView Bridge dispatcher to a database server for the target database. AutoBridge uses tables to identify the data that will go in the records it passes to Tivoli Information Management for z/OS. You specify the data and its format using AutoBridge’s process and mapping tables.

AutoBridge also employs a PostProcessor facility that fills Tivoli Information Management for z/OS records with data not supplied in the alerts, messages, or application data used to create or update the records. The PostProcessor submits the AutoBridge-created records to the same processes (Tivoli Information Management for z/OS user exits, control panels, TSPs, and user-written exit programs) as those records that operators enter using the Tivoli Information Management for z/OS panels at your site.

### Invoking User Functions

AutoBridge can invoke user-written functions as part of its processing. User-written functions can modify or supplement the record data that AutoBridge sends to Tivoli Information Management for z/OS.

## Network Management

Network management includes the tasks that allow an installation to identify, respond to, and track one or more network events simultaneously. Network operators monitor and control the elements of a network by viewing or using resources such as the NetView program and individual network management systems.

Tivoli Information Management for z/OS records are created as the result of someone reporting a failure, or of resource monitoring by the network operators. Network operators use the Tivoli Information Management for z/OS database to track events from their identification to resolution. Accurate information in the network management system is required to evaluate the level of service provided to users and to initiate preventive maintenance based on trend analysis.

The tasks of network management are complicated by the following conditions:

- Manual entry of Tivoli Information Management for z/OS records introduces the possibility of input errors.
- Because it takes longer to enter records manually, personnel have less time to actually respond to the network events.
- When the information entered into Tivoli Information Management for z/OS is delayed or inaccurate, duplicate events are less likely to be identified. Efficiency and productivity are reduced when several support personnel respond to the same event.
- When monitoring multiple systems, there is the question of which system to monitor first. Also, it is not always clear who might be responding to each event. Again, this can reduce efficiency because more than one person might respond to the same event.
- When several events are reported on a network, they are not always entered in Tivoli Information Management for z/OS. For those events that are entered, the time period recorded for identifying and resolving the events might not be accurate. Correct time information is necessary for understanding the number of events occurring in the network and the time required to respond to these events. Without correct time information, you cannot ensure that user needs are being met.

The following sections describe how implementing AutoBridge can help you solve these problems.

## Automating Network Management

AutoBridge can automatically create or update Tivoli Information Management for z/OS records from NetView alerts, messages, and application data. AutoBridge replaces the human action required to detect and enter problems into Tivoli Information Management for z/OS. With AutoBridge, you can customize the record data to meet your operational requirements.

AutoBridge performs some of the network management activities that network operators traditionally perform. These activities include:

- Monitoring for specific alerts, messages, and application data
- Searching for duplicate events
- Creating or updating Tivoli Information Management for z/OS records.

AutoBridge eliminates the administrative time required to perform these activities. Also, because manual actions are eliminated, the record always contains accurate and timely data.

After a record is opened in Tivoli Information Management for z/OS, it can be assigned to a particular individual using the functions of Tivoli Information Management for z/OS, thereby preventing the inefficiency of having several people respond to the same event. Tivoli Information Management for z/OS also allows you to display events by priority, to notify interested parties, and to escalate an event.

AutoBridge eliminates the need to have operators monitor NetView for network events. However, once the event has been identified to the network operator through Tivoli Information Management for z/OS, the operator might need to access NetView to obtain status information, perform network operator functions, and review historical data for additional problem determination.

## Implementation Benefits

The benefits of implementing AutoBridge include:

- **Unattended operation.** In the normal daily activities of a control center, AutoBridge runs unattended. AutoBridge's checkpoint-manager task enables recovery and retry of failed Tivoli Information Management for z/OS transactions, so there is no need to monitor its functioning.

  If you automate procedures, make sure to document those procedures in the operational procedures manual for the site. The operator needs to be aware of automated procedures to prevent duplicating efforts.
- **Eliminates administrative time spent copying information from NetView to Tivoli Information Management for z/OS.** Because data is entered automatically from NetView into Tivoli Information Management for z/OS, network operators do not have to spend time copying this information from one system to the next. AutoBridge also ensures that the data entered is accurate, complete, and timely.
- **Improves management tracking of network activity.** AutoBridge ensures that events identified by NetView are tracked in Tivoli Information Management for z/OS. Maintaining an accurate history of outages helps you identify unstable areas within the network. The result is increased resource efficiency because recurring problems are identified and actions taken to eliminate them. Also, you can track the service you provide, which helps in providing the desired service level.
- **Reduces resources needed to monitor NetView.** AutoBridge sends data directly to Tivoli Information Management for z/OS and identifies any duplicate events. You are provided with one work queue rather than many. Therefore, you no longer need to monitor NetView closely.
- **Reduces down time.** Users should find a higher level of availability in the system as a result of AutoBridge implementation. Alerts, messages, and application data that have been identified to create Tivoli Information Management for z/OS records are passed immediately to Tivoli Information Management for z/OS. The operators begin responding to events immediately rather than waiting for a user to call. Even if the network operators monitor NetView, AutoBridge ensures that all events are identified and tracked.
- **Consistent with manual entry.** AutoBridge's PostProcessor facility allows Tivoli Information Management for z/OS TSPs and exit programs to run in the same manner as they do for records entered manually by operators.

Other less measurable productivity enhancements from implementing AutoBridge include the capability to:
- Prioritize the workload more efficiently
- Estimate future workload needs more accurately based on a complete history of network activity.

AutoBridge allows for a more organized working environment. Events can be displayed in a work queue in priority order so that no confusion exists over which event to respond to next. Also, events can be assigned to specific individuals, further reducing confusion and eliminating duplicate effort. Finally, enough historical data is present in Tivoli Information Management for z/OS to determine when problems occur most often.

# 7 Functional Description of NetView AutoBridge

This chapter describes the various components of AutoBridge and how they interact with each other, with the NetView Bridge, and with the Tivoli Information Management for z/OS NetView Bridge Adapter. NetView Bridge and AutoBridge components include the following:

**NetView Bridge**
A set of APIs that allow the connection of an MVS NetView to external databases or transaction processors. Remote access to the NetView Bridge is also available from the OS/390 system via the remote dispatcher.

**Tivoli Information Management for z/OS NetView Bridge adapter**
An MVS-started task that provides the connection between the NetView Bridge and the Tivoli Information Management for z/OS database. The Tivoli Information Management for z/OS NetView Bridge Adapter works with the NetView Bridge to allow automated transaction-handling functions. The transaction-handling functions consist of message routing and transmission within the Tivoli Information Management for z/OS NetView Bridge Adapter address space and message processing and submission to the Tivoli Information Management for z/OS database. Also known as the *database server adapter*.

**Bridge dispatcher**
A NetView autotask that enables transactions to be passed between a NetView application and a resident database server.

**Remote dispatcher (Optional)**
A NetView autotask that enables transactions to be passed between a command procedure on a remote host and an external database on the resident OS/390 host. Two remote dispatchers are required, one on the resident OS/390 host and one on the remote system.

**Checkpoint task**
An autotask provided by AutoBridge that logs each AutoBridge transaction along with a code indicating its current status (waiting, ready for resend, failed, or flagged for deletion) to prevent the loss of transactions and to allow retries of failed transactions.

**PostProcessor tasks (Optional)**
AutoBridge's PostProcessor facility resides on the system as a set of background tasks, one for each Tivoli Information Management for z/OS NetView Bridge Adapter. There can be as many as 16 PostProcessor tasks on the resident NetView. When a PostProcessor task detects an AutoBridge transaction occurring on its

associated adapter, it initiates post-processing of AutoBridge-created records in the Tivoli Information Management for z/OS database.

## AutoBridge and NetView Bridge Components

AutoBridge resides in the resident NetView and, optionally, in remote NetViews. The resident NetView is the OS/390 host with the Tivoli Information Management for z/OS NetView Bridge Adapter and the target database. To AutoBridge, all NetViews that are not co-resident with the target database are considered remote.

Figure 3 shows a diagram of the NetView Bridge and AutoBridge components on both remote and resident NetViews.

The following sections explain the numbered paths in Figure 3. The numbers in the figures show NetView Bridge actions, while the letters show AutoBridge actions.

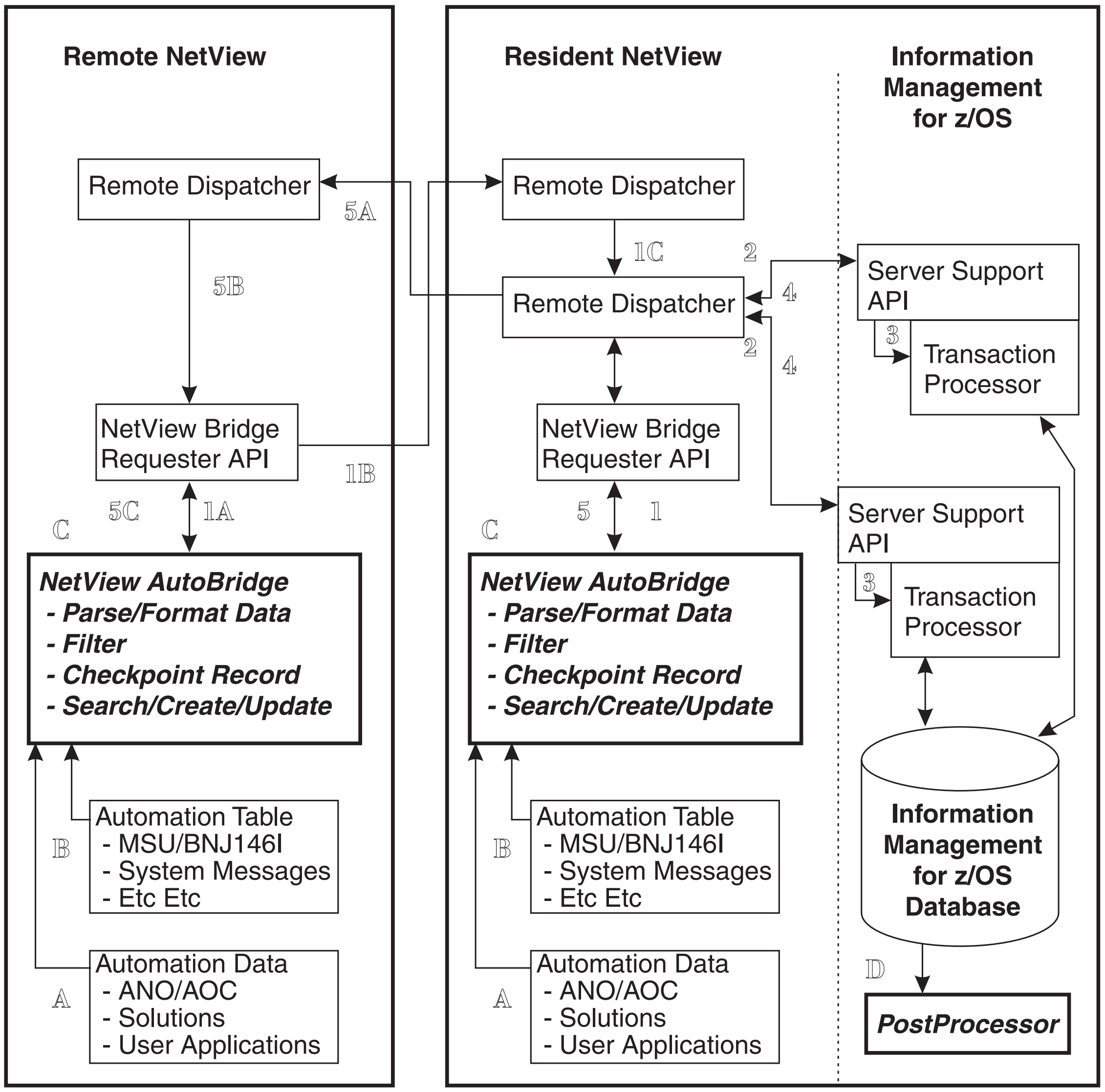


Figure 3. Interaction of NetView Bridge and AutoBridge components

## NetView Bridge on the Resident Host

**1** AutoBridge uses the NetView Bridge Requester API to request that a search, create, or update transaction request be sent to a database server.

**2** The bridge dispatcher passes this request to an available database server.

**3** The Tivoli Information Management for z/OS NetView Bridge Adapter is a database server which uses the server support API to retrieve the transaction. The server performs the requested transaction by invoking the HLAPI, which works directly with the Tivoli Information Management for z/OS database.

**4** The Tivoli Information Management for z/OS NetView Bridge Adapter creates a reply for this request. It uses the server support API to pass the reply back to AutoBridge through the bridge dispatcher in the form of a message.

**5** AutoBridge uses the NetView Bridge Requester API to extract the reply data from the message.

## AutoBridge on the Resident Host

**A** An application running in NetView invokes AutoBridge to retrieve or send a record to the Tivoli Information Management for z/OS database.

**B** An MSU or message causes the NetView automation table to invoke AutoBridge to retrieve or send a record to the Tivoli Information Management for z/OS database.

**C** The input parameters specify what processing is required. This can include parsing and filtering of the input data and record retrieval or create/update.

**D** Once the record has been created, it may be post-processed to cause all user-selected TSPs and exits to execute.

## NetView Bridge on a Remote Host

**1A** AutoBridge uses the NetView Bridge Requestor API to request that a search, create, or update transaction request be sent to the remote dispatcher.

**1B** The high performance transport API acts as an interface between the remote NetView and the resident NetView to send the request to the remote dispatcher in the resident NetView.

**1C** The remote dispatcher sends the request to the bridge dispatcher running in the resident NetView.

**3** The bridge dispatcher passes this request to an available database server.

**3** The Tivoli Information Management for z/OS NetView Bridge Adapter is a database server which uses the server support API to retrieve the transaction. The server performs the requested transaction by invoking the HLAPI, which works directly with the Tivoli Information Management for z/OS database.

**4** The Tivoli Information Management for z/OS NetView Bridge Adapter creates a reply for this request, it uses the server support API to pass the reply back to AutoBridge through the bridge dispatcher in the form of a message.

**5A** The bridge dispatcher uses the high performance transport API to send the reply to the remote dispatcher on the remote NetView system.

**5B** The remote dispatcher sends the reply to the NetView Bridge Requester API.

**5C** AutoBridge uses the NetView Bridge Requestor API to extract the reply data from the message.

### AutoBridge on a Remote Host

**A** An application running in NetView invokes AutoBridge to retrieve or send a record to the Tivoli Information Management for z/OS database.

**B** An MSU or message causes the NetView automation table to invoke AutoBridge to retrieve or send a record to the Tivoli Information Management for z/OS database.

**C** The input parameters specify what processing is required. This may include parsing and filtering of the input data and record retrieval or create/update.

## Processing Overview

AutoBridge is invoked in one of two ways:

- The NetView automation table invokes AutoBridge in response to certain alerts and messages.
- Other applications, routines, or command processors request the NetView Bridge via AutoBridge.

AutoBridge receives input records from the NetView automation table as parameters in the form of MSUs, system messages, and other application messages. Application programs can pass data to AutoBridge in the form of a parameter list.

Figure 4 on page 77 shows a diagram of the major tasks performed by AutoBridge to process a transaction record. An explanation of this process follows the figure.

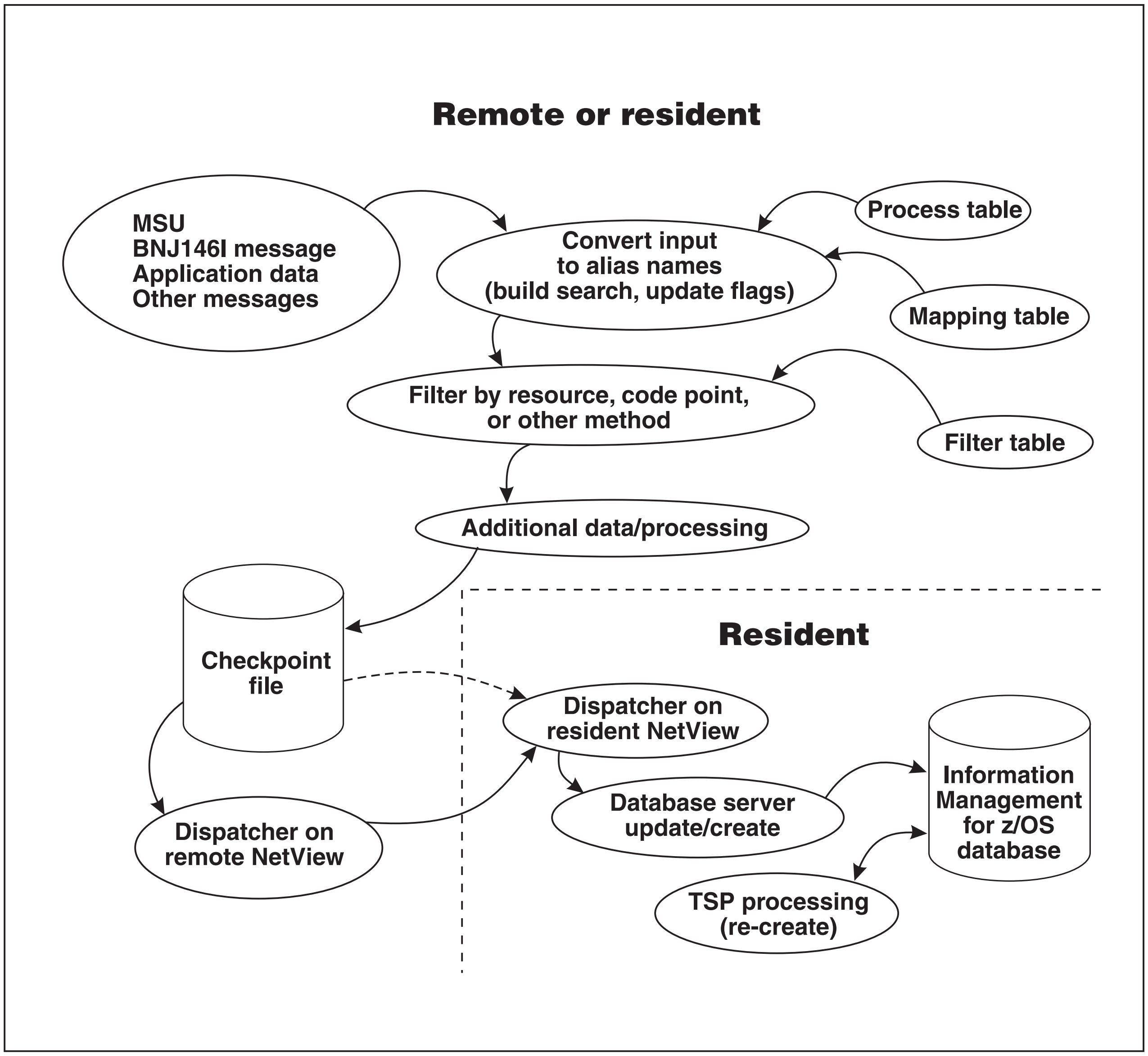

Remote or resident
MSU
BNJ146I message
Application data
Other messages
Convert input to alias names (build search, update flags)
Process table
Mapping table
Filter by resource, code point, or other method
Filter table
Additional data/processing
Checkpoint file
Resident
Dispatcher on resident NetView
Dispatcher on remote NetView
Database server update/create
Information Management for z/OS database
TSP processing (re-create)

*Figure 4. AutoBridge process flow*

The major processing tasks of AutoBridge are:

1. Entries in the NetView automation table that respond to MSUs or messages can drive AutoBridge, or an application program (such as an automation or expert system) can pass data to AutoBridge in the form of a parameter list. When the NetView automation table entry or application invokes AutoBridge, the process table name, NetView Bridge dispatcher name, and input data name are passed as parameters.
2. The steps specified in the process table are performed including parsing input data with a mapping table to form a transaction record.
3. Any additional processing or data gathering is performed.
4. The transaction record is saved in the checkpoint file on the current NetView. This allows for error recovery retries.

5. The transaction record is forwarded to a Tivoli Information Management for z/OS NetView Bridge Adapter database server on the resident NetView. If coming from a remote NetView, the transaction passes through a remote NetView Bridge dispatcher.
6. The Tivoli Information Management for z/OS NetView Bridge Adapter database server performs the transaction (create, update, or search).
7. The transaction record in the checkpoint file is marked successful or unsuccessful. If successful, the transaction record is deleted from the checkpoint file.
8. If unsuccessful, the transaction is retried periodically until the retry count specified in the initialization table is exhausted.
9. The record can be post-processed on the resident NetView to fill Tivoli Information Management for z/OS fields with data not supplied in the transaction record.

## Process Invocation

When AutoBridge receives an input record, it can invoke functions in response to that record as specified in its process table. The process table is stored in the EYLATPRO member of the DSIPARM data set.

The process table is divided into sections called *process segments*. Each segment is uniquely named and contains the set of AutoBridge, user, and NetView commands required to process the specified input record. These functions can be any of those defined in AutoBridge (such as those that create, update, and retrieve records in Tivoli Information Management for z/OS), or they can be user-defined functions.

In addition to input data, calls to AutoBridge include a parameter that specifies the process table segment containing the steps to be performed with that data. When invoked, AutoBridge retrieves this specific segment and steps through the statements in it, executing the function or procedure specified on each statement.

Process segments that call AutoBridge’s PARSE function use AutoBridge’s mapping table and, optionally, filter table. These tables are described in the following sections.

For more information on the process table, see “Coding the Process Table” on page 83.

## Database Mapping

AutoBridge’s mapping table enables you to map the contents of input records captured by AutoBridge to specific fields in the Tivoli Information Management for z/OS database. The mapping table is stored in the EYLATMAP member of the DSIPARM data set.

The mapping table’s primary function is to parse the input data. As in the process table, mapping table statements are grouped into unique and identifiable segments. The process table’s PARSE function specifies a segment in the mapping table that contains the statements required to map the input from specific alerts, messages, or application data to specific fields in the Tivoli Information Management for z/OS database.

Because each process table segment can contain multiple parsing statements, each segment can use multiple segments in the mapping table to parse the input data.

For more information on the mapping table, see “Coding the Mapping Table” on page 92.

## Input Record Filtering

AutoBridge's filter table enables AutoBridge to filter input records. AutoBridge can invoke this internal filtering on the PARSE statement in the process table, so that input records can be filtered after parsing. When this filtering is specified, AutoBridge compares the parsed input record against user-specified filter values to determine whether it should be passed on or blocked from further processing.

The filter table has a default value that can be set to PASS or BLOCK, allowing you to specify whether input records that match the filter values are passed or blocked. The filter table is stored in the EYLATFIL member of the DSIPARM data set.

See "Coding the Filter Table" on page 97 for more information.

## Checkpoint Management

AutoBridge provides transaction logging and recovery through its checkpoint manager function, which provides the ability to retry failed or incomplete AutoBridge transactions.

Each AutoBridge transaction is logged by a specific checkpoint manager associated with the dispatcher specified on AutoBridge invocation. In the event of failure (such as a record is in use–being updated– by another Tivoli Information Management for z/OS user), the transaction is retrieved by the checkpoint manager and retransmitted. The time interval between retransmissions is specified on the initialization table's RETRYINT setting. AutoBridge will retry the transmission as many times as is specified on the initialization table's RETRYNUM setting. If none of these retry attempts are successful, AutoBridge will flag the transaction as failed, place a *transaction failed* message in the NetView log, and make no further attempts to resend the transaction.

Most failed transactions are caused by incompatibility of the transaction data with Tivoli Information Management for z/OS. For example, transaction records containing incorrect alias names, unsuccessful verifier matches, or logic errors will fail to update Tivoli Information Management for z/OS; therefore, the checkpoint manager will mark such transactions as failed.

The checkpoint manager compensates for one type of transaction error. If a conditional create transaction fails because a duplicate record already exists in Tivoli Information Management for z/OS, the checkpoint manager will convert the transaction to an update transaction and submit it in place of the failed create transaction.

When a transaction completes successfully, the checkpoint manager processes the transaction response from the NetView dispatcher and returns this response to the submitter in the form of a multiline message.

## Transaction Post-Processing

AutoBridge's PostProcessor facility locates and processes AutoBridge-created records in the Tivoli Information Management for z/OS database. The PostProcessor subjects these records to the same processes (Tivoli Information Management for z/OS user exits, control panels, TSPs and user-written exit programs) as those records that are manually entered by operators through Tivoli Information Management for z/OS panels supplied by IBM® and other vendors and those modified at your site. This post-processing is necessary because the records created by AutoBridge do not flow through the interactive fields and selections that trigger these additional processes, many of which supplement or modify the record data. Figure 5 shows a representation of the data flow in PostProcessor record processing.

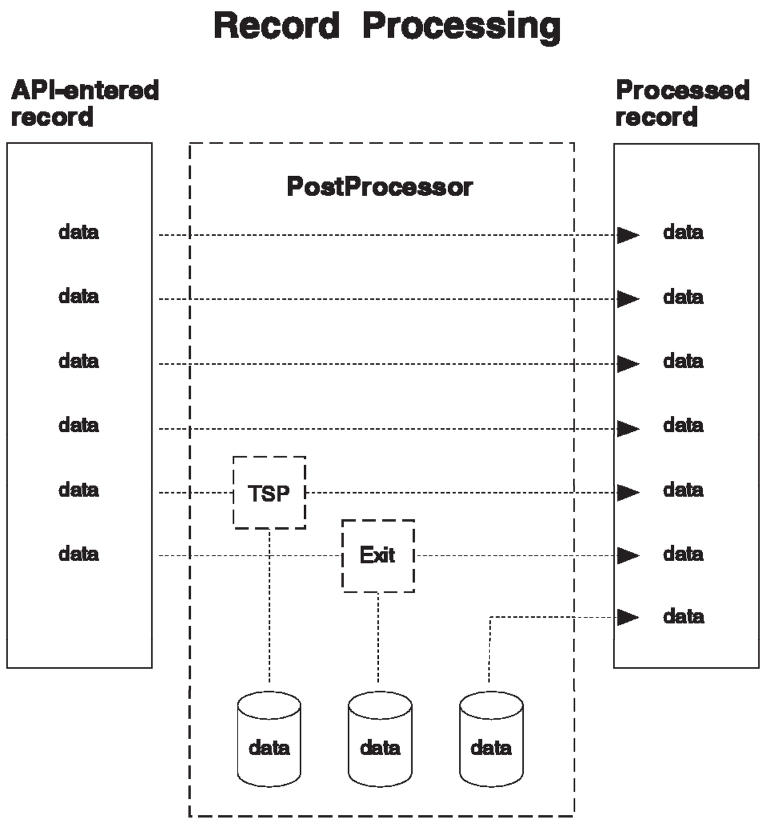


*Figure 5. PostProcessor record processing*

The PostProcessor has two major functions. Its first function is to detect the creation of a record by AutoBridge. Its second function is to perform the actual processing of the record.

When the PostProcessor finds a record created by AutoBridge, it processes the record based on customer-defined parameters specified in a new type of Tivoli Information Management for z/OS record called a *mapping reference record*. The mapping reference record contains entries corresponding to the Tivoli Information Management for z/OS panel items that cause modification of the record data. Using the mapping reference record, the PostProcessor performs the same actions as would occur if an operator entered the data manually. After the record is successfully completed and filed, the original AutoBridge-created record is deleted.

Use of the PostProcessor is optional. For more information on the PostProcessor, see "Using the NetView AutoBridge PostProcessor" on page 163.

8

# Coding NetView AutoBridge Tables

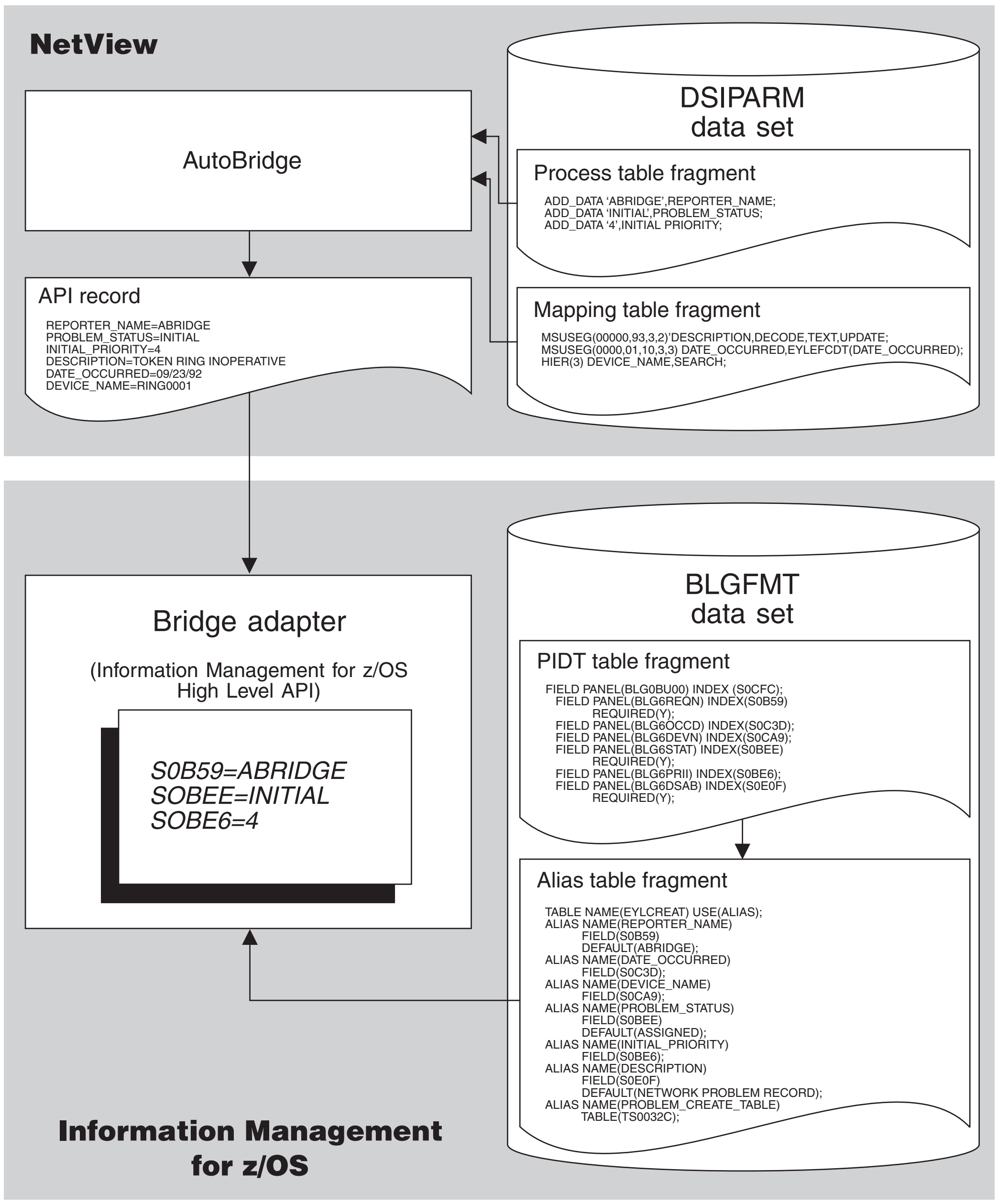
This chapter provides the following information on coding AutoBridge's various tables:

- "Coding the Process Table" on page 83
- "Coding the Filter Table" on page 97
- "Coding the Mapping Table" on page 92
- "Coding the Initialization Table" on page 101.

"AutoBridge Table Examples" on page 98 provides examples of AutoBridge's process, mapping, and filter tables and describes how they respond and relate to AutoBridge API invocations for processing generic alerts and BNJ146I messages.

Figure 6 on page 82 shows how AutoBridge uses the process and mapping tables to assign input data to a panel field.

An example of the initialization table is shown in Figure 8 on page 103.



```
ADD_DATA 'ABRIDGE',REPORTER_NAME;
ADD_DATA 'INITIAL',PROBLEM_STATUS;
ADD_DATA '4',INITIAL PRIORITY;
```

```
MSUSEG(00000,93,3,2)'DESCRIPTION,DECODE,TEXT,UPDATE;
MSUSEG(0000,01,10,3,3) DATE_OCCURRED,EYLEFCDT(DATE_OCCURRED);
HIER(3) DEVICE_NAME,SEARCH;
```

```
REPORTER_NAME=ABRIDGE
PROBLEM_STATUS=INITIAL
INITIAL_PRIORITY=4
DESCRIPTION=TOKEN RING INOPERATIVE
DATE_OCCURRED=09/23/92
DEVICE_NAME=RING0001
```

```
S0B59=ABRIDGE
SOBEE=INITIAL
SOBE6=4
```

```
FIELD PANEL(BLG0BU00) INDEX (S0CFC);
  FIELD PANEL(BLG6REQN) INDEX(S0B59)
          REQUIRED(Y);
  FIELD PANEL(BLG6OCCD) INDEX(S0C3D);
  FIELD PANEL(BLG6DEVN) INDEX(S0CA9);
  FIELD PANEL(BLG6STAT) INDEX(S0BEE)
          REQUIRED(Y);
  FIELD PANEL(BLG6PRII) INDEX(S0BE6);
  FIELD PANEL(BLG6DSAB) INDEX(S0E0F)
          REQUIRED(Y);
```

```
TABLE NAME(EYLCREAT) USE(ALIAS);
ALIAS NAME(REPORTER_NAME)
         FIELD(S0B59)
         DEFAULT(ABRIDGE);
ALIAS NAME(DATE_OCCURRED)
         FIELD(S0C3D);
ALIAS NAME(DEVICE_NAME)
         FIELD(S0CA9);
ALIAS NAME(PROBLEM_STATUS)
         FIELD(S0BEE)
         DEFAULT(ASSIGNED);
ALIAS NAME(INITIAL_PRIORITY)
         FIELD(S0BE6);
ALIAS NAME(DESCRIPTION)
         FIELD(S0E0F)
         DEFAULT(NETWORK PROBLEM RECORD);
ALIAS NAME(PROBLEM_CREATE_TABLE)
         TABLE(TS0032C);
```

*Figure 6. Data mapping from AutoBridge to Tivoli Information Management for z/OS*

The Tivoli Information Management for z/OS API defines the data it handles using Program Interface Data Tables (PIDTs) and Program Interface Alias Tables (PALTs, better known as *alias tables*). PIDTs are required to define the data that is passed to the Tivoli Information Management for z/OS API. In a PIDT, panel fields are defined in terms of their structured word (s-word) and prefix word (p-word) indexing and by whether the field is required. Additional information on PIDTs, PALTs, and data view records can be found in the *Tivoli Information Management for z/OS Application Program Interface Guide*.

The alias table assigns identifying names, or *alias names*, to the s-word definitions of the fields specified in the PIDTs. A field's alias definition in the alias table includes an alias name (usually describing the contents of the field), the associated s-word specified in the PIDT, and may also have a default value that the field will contain if no input is provided.

Once these tables are defined, the Tivoli Information Management for z/OS NetView Bridge Adapter uses them as its interface to the database.

In the process, mapping, and filter tables, the value in each *to_name* parameter is the alias name or s-word of the field that is the destination of the input data.

## Coding the Process Table

The process table consists of statements that are grouped into segments. Each segment is delimited by a BEGIN and an END statement. Statements that are not in a segment cannot be referenced and therefore will never be processed.

This is an example of a process table segment.

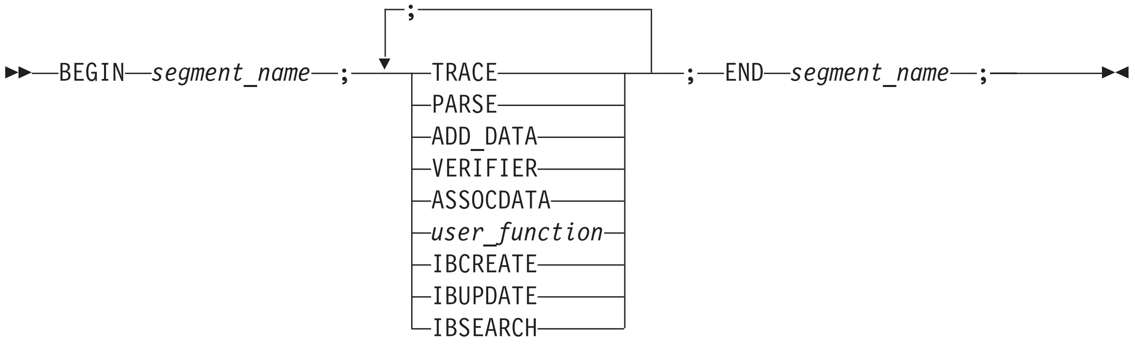
```
****  PROCESS TABLE SEGMENT FOR BNJ146I MESSAGE: NON-NMVT(R) FORMAT
*
BEGIN PROCESS_BNJ146R;
ADD_DATA 'INITIAL',PROBLEM_STATUS;
ADD_DATA 'NETVIEW',REPORTER_NAME;
ADD_DATA 'D15A',REPORTER_DEPT;
ADD_DATA '3',INITIAL_PRIORITY;
ADD_DATA '555-5430',REPORTER_PHONE;
ADD_DATA 'AUTOBRG',PROGRAM_NAME;
PARSE MAPPING=MAP_BNJ146R,FILTER;
IBCREATE 'INFONETW','PROBLEM';
END PROCESS_BNJ146R;
*
*
```

Comments in the process table cannot share a line with other statements and must be prefixed by one or more asterisks (*) beginning in column one. All non-comment lines must end with a semicolon (;).

The process table resides in the EYLATPRO member of the DSIPARM data set. Process table segments must reside either in the EYLATPRO member or in another member of DSIPARM that is referred to from the process table by an %INCLUDE statement.

The functions supported in process table statements are described in the following sections. You may extend the capabilities of the process table by developing and adding your own functions to AutoBridge.

The general format of the process table statement is:



where:

**BEGIN**
Marks the beginning of a segment within the process table.

*segment_name*
Identifies the segment and must be present for each segment in the process table. The segment name must be identified on a segment's BEGIN and END statements. The segment name can be 1—32 characters.

**END**
Marks the end of a segment within the process table. Each segment must end with an END statement.

The following sections describe each of the process table functions.

## TRACE Function Syntax

```
                                          ┌─ALL──────────────────────────┐
                                          │  ┌─,─────────────────────┐   │
                           ┌─MOD──┐       │  ▼                       │   │
►►──TRACE──┬─ON──┬─,──┼─DATA─┼──,──┴──┴─┬───────────────┬─┴───┴─;──────────────►◄
           └─OFF─┘    ├─REXX─┤          └─function_name─┘
                      └─ALL──┘
```

where:

**TRACE**
Invokes AutoBridge's tracing function.

**ON**
Starts tracing from the point of invocation.

**OFF**
Stops tracing if it was active.

**MOD**
Enables or disables trace statements for entry to and exit from the named functions. If no function name is specified, entry and exit tracing is enabled or disabled for all functions. MOD is the default.

**DATA**
Enables or disables the tracing of transaction data through one or more functional components.

**REXX**
Enables or disables the REXX language TRACE intermediate instruction. This option applies only to REXX program modules.

**ALL**
Enables or disables all active or inactive trace options. For example, consider the following TRACE command:

```
TRACE OFF,ALL
```

If one or more trace options were active, this command turns off all tracing for all functions.

*function_name*
Sets tracing for the named function. One or more functions may be specified.

The following function abbreviations are supported for this command:

| | | | |
|---|---|---|---|
| **TM** | Table manager | **CP** | Checkpoint manager |
| **PT** | Process table | **HLM** | High level manager |
| **API** | AutoBridge API | | |

**ALL**
Indicates that tracing will be set for all AutoBridge functions. This is the default.

Following are examples of TRACE statements:

```
TRACE ON,MOD,TM,CP;
```

This trace statement causes module entry and exit tracing to be invoked for the table manager and checkpoint manager functions.

```
TRACE ON,DATA,ALL;
```

This trace statement causes DATA level tracing to be invoked for all traceable functions in AutoBridge, including the table manager, process table, API, checkpoint manager, and high-level manager functions.

## PARSE Function Syntax

```
►►──PARSE──MAPPING=segment_name──┬─────────┬──;──────────────────────────────►◄
                                 └─,FILTER─┘
```

where:

**PARSE**
Parses the input using the specified segment from the mapping table. The parsing instructions specified in the segment are performed in sequence.

**MAPPING=***segment_name*
Is the name of the segment in the mapping table that is to be used to parse the input record. One or more statements may be contained in a mapping table segment.

**FILTER**
Specifies that AutoBridge filtering is to be performed after the mapping segment is processed.

**Note:** For best performance, you should filter input records before invoking AutoBridge. If, however, you choose to use AutoBridge's filtering capability, then you should note that filtering for a specific PARSE statement is not performed until all statements in the specified mapping table segment have processed. Use separate PARSE statements to parse data that is being filtered to avoid unnecessary parsing of data that will be blocked by the filter table.

Use the following conventions in process table segments containing multiple PARSE statements:

- Structure multiple PARSE statements in your process table segments so that the data that you are filtering is parsed first.
- Specify the FILTER option before parsing any non-filtered data to prevent unnecessary processing.

The following is an example of a PARSE statement:

```
PARSE MAPPING=MAP_GENALERT1,FILTER;
```

The mapping keyword identifies the mapping table segment (MAP_GENALERT1) used to parse the input data. The FILTER keyword invokes AutoBridge filtering.

## ADD_DATA Function Syntax

```
                                          ┌─,────────────┐
                                          ▼              │
►►──ADD_DATA──from_field──,──to_name──,────┬─TEXT───┬───┴──;──────────────────────►◄
                                           ├─SEARCH─┤
                                           └─UPDATE─┘
```

where:

**ADD_DATA**
Allows data that is not part of the input to be added to the transaction record.

*from_field*
Specifies the source of the data that is to become part of the transaction. This field may be a variable, a literal string (enclosed in quotes), or a function that will create the data as its result.

The *from_field* parameter may be defined in different ways as illustrated below:

- As a literal string: `ADD_DATA 'OPEN',PROBLEM_STATUS;`
- As a NetView task global variable: `ADD_DATA STATUS,PROBLEM_STATUS;`
  The variable STATUS is retrieved and its value assigned to PROBLEM_STATUS.
- As a NetView task global stemmed variable: `ADD_DATA SER_ADDR,ADDRESS_TEXT,TEXT;`
  Data must be passed in a stemmed variable if the data is more than 255 characters.
- As a REXX built-in or user-written function. The result of the function is saved in the alias name or s-word. Here are two examples of the exec_name parameter:

  ```
  ADD_DATA DATE(U),DATE_OCCURRED,UPDATE;
  ADD_DATA NVID(),NETWORK_NAME;
  ```

If the *from_field* value represents a stemmed variable containing freeform text that is to be added to the TEXTLIST parameter for creation or update, the following restrictions apply:

- The *from_field.0* value must be numeric. It represents the number of stemmed variables or lines that make up the text.
- The stem tail must be numeric and in ascending order, with the first stem (*from_field.1*) being the first line of the text, *from_field.2* the second line, and so on.
- Text may be added to a previously created stemmed variable that has been added to the TEXTLIST parameter for a create or update transaction.

See page 114 for a description of stemmed variables.

*to_name*
: Is the alias name or s-word assigned to the field in the transaction record where the input data is to be stored.

**TEXT**
: Indicates that the *to_name* variable is to be appended to the variable TEXTLIST of the transaction record as freeform text.

  See "Coding the Mapping Table" on page 92 for more information on the function of this parameter.

**SEARCH**
: Indicates that the *to_name* variable is added to the search list of any search transaction that may be created for this input record.

  See "Coding the Mapping Table" on page 92 for more information on the function of this parameter.

**UPDATE**
: Indicates that the *to_name* variable is appended to the update list of the update transaction.

  See "Coding the Mapping Table" on page 92 for more information on the function of this parameter.

## VERIFIER Function Syntax

```
►►──VERIFIER──parm=value──┬──────────────────────────────┬──;──────────────►◄
                          └─,parm=value,...,parm=value───┘
```

where:

**VERIFIER**
: Is the NetView Bridge keyword that is used in an update transaction to verify the record being updated.

*parm=value*
: Identifies one or more field/value pairs that must be checked against the matching database record as a means of verification prior to updating the record.

The following is an example of a VERIFIER statement:

```
VERIFIER REPORTER_NAME=AUTOBRIDGE,PROBLEM_STATUS=INITIAL;
```

The two entries on this statement, REPORTER_NAME and PROBLEM_STATUS, are used to perform a verification on the record selected for update. That is, the value of REPORTER_NAME must be AUTOBRIDGE and that of PROBLEM_STATUS must be INITIAL or the record will not be updated.

## ASSOCDATA Function Syntax

```
►►──ASSOCDATA──=──name──────────────────────────────────────────────────►◄
```

where:

**ASSOCDATA**
Is the NetView Bridge keyword that is used in a create, conditional update, or search transaction to identify an alias name to contain additional data returned with the search data.

*name*
Is the alias name or s-word of the field in a database record whose contents are returned if the record matches the search criteria.

The value of this field is returned on each of the EYL553I messages that result from a transaction that performs a search, along with the record numbers located by the search.

## User Function or Command Invocation Syntax

```
►►──target_var=─function_name(──┬────────────────┬──)──;──────────────────────────►◄
                                └─parm,...,parm──┘
```

or

```
►►──command──;──────────────────────────────────────────────────────────────────►◄
```

where:

*target_var*
Is the variable to contain the results of the function.

*function_name*
Is the built-in REXX function or user-written function to be called.

*(parm,...,parm)*
Are the parameters passed to the function. Depending on the function, there may be no parameters. If there are no parameters, use empty parentheses to specify a null entry.

*command*
Is the name of any valid command, command list, or command processor. The *command* value is invoked as if entered on the command line.

The following is an example of the syntax for invoking functions:

```
  ADD_DATA SUBSTR(TIME(),1,5),TIME_OCCURRED;
```

In this example, the target variable TIME_OCCURRED is set to characters 1 to 5 of the function TIME and so will return a value in the format *HH:MM* such as 12:36.

Here are two examples of the invocation syntax for issuing commands:

```
  GLOBALV GETC RECID;
  ADD_DATA recid,RECORDID;
-- or --
  MSG MYID,I AM DEBUGGING PROCESS LIST 12
```

In the first example, the NetView GLOBALV command is called to return a common global value of RECID, which is then added to the transaction as RECORDID.

In the second example, the NetView MSG command is called to issue a message to operator MYID with the text “I AM DEBUGGING PROCESS LIST 12”.

## IBCREATE Function Syntax

```
►►──IBCREATE──target_database──,──record_type──;─────────────────────►◄
```

where:

**IBCREATE**
Identifies the TRANSID of the transaction that is to be generated. The IBCREATE function causes a transaction to be built that creates a new record in the target database. The IBCREATE function is conditional if any of the mapping table statements used to parse this data contain the keyword SEARCH. If the SEARCH keyword is used and a record already exists in the database having field values matching those of the transaction record, then a new record is not created. Instead, the existing record is updated. If two or more records are found, the update is performed on the record with the highest record value on the results list returned by Tivoli Information Management for z/OS. If the ASSOCDATA keyword is specified in the process table, the value of the alias name or s-word specified in the ASSOCDATA statement is also returned.

*target_database*
Is a literal or variable containing the label of the database (as defined in the supplied initialization table) where the transaction is to be sent. In the initialization table, the Tivoli Information Management for z/OS database is registered as INFONETW.

*record_type*
Is a literal or variable containing the label for this type of record as defined in the initialization table.

The following is an example of an IBCREATE statement:

```
  IBCREATE 'INFONETW','PROBLEM';
```

This statement results in a create transaction of type PROBLEM being sent over the NetView Bridge to the Tivoli Information Management for z/OS database INFONETW. The initialization table can have more than one registered database; each database must have a unique name.

If you would like to use data view records with the IBCREATE, specify in the process table

```
ADD_DATA 'YES',USE_DATA_VIEW;
```

## IBUPDATE Function Syntax

```
►►──IBUPDATE──target_database──,──record_type──;─────────────────────►◄
```

where:

**IBUPDATE**
Identifies the TRANSID of the transaction that is to be generated. IBUPDATE causes a transaction to be built that will update a specified record in the target database. If the VERIFIER keyword is specified in the process table, then the update is only performed when each VERIFIER statement has been satisfied.

*target_database*
: Is a literal or variable containing the label of the database (as defined in the initialization table) where the transaction is to be sent.

*record_type*
: Is a literal or variable containing the label for this type of record as defined in the initialization table.

If you know the ID of the record you want to update, specify the RECORDID in the process or mapping table. The checkpoint manager will send the update request to the bridge adapter. Specify it in the process table as:

```
ADD_DATA my_rec_id,RECORDID;  my_rec_id is a task global = a recid
-- or --
ADD_DATA '00347001',RECORDID;
```

or specify it in the mapping table as:

```
my_rec_id(1,1) RECORDID;
```

If you don't know the ID of the record to be updated, you can call the AutoBridge API without a RECORDID. However, search arguments must be defined in the process or mapping table statements (that is, one or more statements must include ',SEARCH'). The checkpoint manager will process the request as an IBSEARCH, and update the record with the highest RECORDID value retrieved. If no records are found that meet search criteria, no record is updated and the transaction is deleted.

The following is an example of an IBUPDATE statement:

```
IBUPDATE 'INFONETW','PROBLEM';
```

This statement results in an update transaction of type PROBLEM being sent over the NetView Bridge to the Tivoli Information Management for z/OS database INFONETW.

If you would like to replace existing freeform text using IBUPDATE, specify in the process table

```
ADD_DATA 'YES',REPLACE_TEXT_DATA;
```

If you would like to use data view records with the IBUPDATE, specify in the process table

```
ADD_DATA 'YES',USE_DATA_VIEW;
```

## IBSEARCH Function Syntax

```
▶▶──IBSEARCH──target_database──,──record-type──;──────────────────────────▶◀
```

where:

**IBSEARCH**
: Identifies the TRANSID of the transaction to be generated. IBSEARCH causes a transaction to be built that will search for one or more records in the target database. This transaction ID requires that the fields to be searched be listed in the NetView Bridge control variable SEARCHLIST. Do this by using the SEARCH keyword on an ADD_DATA function in the process table or on an entry in the mapping table.

*target_database*
Is a literal or variable containing the label of the database (as defined in the initialization table) where the transaction is to be sent.

*record_type*
Is a literal or variable containing the label for this type of record as defined in the initialization table.

The following is an example of an IBSEARCH statement:

```
IBSEARCH 'INFONETW','PROBLEM';
```

This statement results in a search transaction of type PROBLEM being sent over the NetView Bridge to the target Tivoli Information Management for z/OS database INFONETW.

There are three types of search transactions, which specify the following types of search:

**Search with record id**
Parmvar contains the record id only. For example,

```
ADD_DATA '00347001',RECORDID,SEARCH;
```

The fields defined in the Tivoli Information Management for z/OS Retrieve PIDT for the specified record are returned. EYL553I is returned in the form:

```
EYL553I alias|s-word IS value

EYL567I TRANSACTION IBSEARCH ABR10@D3 COMPLETED SUCCESSFULLY
EYL553I S0032 IS RECS=PROBLEM
EYL553I S0CFC IS Reporter data
EYL553I REPORTER_NAME IS KATHERINE
EYL553I NETWORK_NAME IS CNM01
EYL553I DEVICE_NAME IS LANB01E2
EYL553I PROBLEM_TYPE IS HARDWARE
EYL553I PROBLEM_STATUS IS OPEN
EYL553I INITIAL_PRIORITY IS 04
EYL553I DESCRIPTION IS LINK CONNECTION FAILED
EYL553I S0CFD IS Status data
EYL553I ASSIGNEE_NAME IS PFDOWNING
EYL553I S0C0B IS INPROG
EYL553I CURRENT_PRIORITY IS 02
      .
      .
      .
EYL554I END OF DATA
```

When using a search with RECORDID, a retrieve (but not a search) is performed; any **ASSOCDATA** is ignored.

**Search with arguments**
Parmvar contains aliases or s-words with search arguments.

For example, if the mapping table contains:

```
MSUSEG(0000.92,5,1) PROBLEM_TYPE,DECODE,SEARCH;
HIER() DEVICE_NAME,EYLEXHIR(N,L,DEVICE_NAME),SEARCH;
```

and the process table contains:

```
ADD_DATA 'OPEN',PROBLEM_STATUS,SEARCH;
IBSEARCH 'INFONETW','PROBLEM';
```

```
-- or --

  ADD_DATA 'OPEN',PROBLEM_STATUS,SEARCH;
  VERIFIER REPORTER_NAME=BRIDGE,PROBLEM_STATUS=ASSIGNED;
  IBSEARCH 'INFONETW','PROBLEM';
```

Assume PROBLEM_TYPE is decoded to 'COMMUNICATIONS' and DEVICE_NAME is 'RING001' and the alias names are the default s-words. The search argument is passed to Tivoli Information Management for z/OS as:

```
SEARCH S0C09=COMMUNICATIONS,S0CA9=RING001,S0BEE=OPEN
```

An array of one or more record ids is returned in the form:

```
EYL553I RESULT IS recordid record_s-word assocdata code

EYL567I TRANSACTION IBSEARCH CNM01@X1 COMPLETED SUCCESSFULLY
EYL553I RESULT IS 00045678 S0032 00
EYL554I END OF DATA
```

**Search with arguments and associated data**

Parmvar contains aliases or s-words with search arguments and one **ASSOCDATA** parameter. For example,

```
ASSOCDATA=REPORTER_NAME
```

An array of one or more record ids and associated data is returned. EYL553I is returned in the form:

```
EYL553I RESULT IS recordid record_s-word assocdata code

EYL567I TRANSACTION IBSEARCH CNM01@X2 COMPLETED SUCCESSFULLY
EYL553I RESULT IS 00045678 S0032 AUTOBRIDGE      00
EYL553I RESULT IS 00045679 S0032 KGSANDERS       00
EYL553I RESULT IS 00045680 S0032 MAHANEY         00
EYL553I RESULT IS 00059999 S0032 LAPOINTE        00
EYL554I END OF DATA
```

The “code” value is defined as follows:

| | |
|---|---|
| **00** | No error detected. |
| **01** | The record encountered a read error. |
| **02** | The record was not found. |
| **03** | The record was not currently available. |
| **04** | The record was currently busy. |
| **05** | Not enough storage to read in the record. |
| **06** | Unknown problem when reading record. |

**Search using data views**

If you would like to use data view records with the IBSEARCH, specify in the process table

```
ADD_DATA 'YES',USE_DATA_VIEW;
```

## Coding the Mapping Table

The mapping table consists of statements that are grouped into segments. Each segment is delimited by a BEGIN and an END statement. Statements that are not in a segment cannot be referenced and therefore will never be executed.

This is an example of a mapping table segment.

```
***** MAPPING SEGMENT FOR MSU FROM LAN MANAGER ****************
BEGIN  LANMGR1;
MSUSEG(0000.92,5,1) DESCRIPTION,DECODE,UPDATE;
MSUSEG(0000.93,3,2) PROBABLE_CAUSE,DECODE,SEARCH;
MSUSEG(0000.01.10,3,3) DATE_OCCURRED,EYLEXCDT('DATE_OCCURRED'X);
MSUSEG(0000.01.10,6,3) TIME_OCCURRED,EYLEXCTM('TIME_OCCURRED'X);
HIER(4) DEVICE_NAME,EYLEXNAM(DEVICE_NAME),SEARCH;
MSUSEG(0000.94.01,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(0000.94.81,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(0000.94.81,5,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(0000.95.01,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(0000.96.01,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(0000.31.30,3) DESCRIPTION_TEXT,TEXT;
MSUSEG(0000.10(1).11(2).06,3) PRODUCT_NAME,UPDATE;
END LANMGR1;
***** END OF MSU MAPPING SEGMENT  *****************************
```

Comments in the mapping table cannot share a line with other statements and must be prefixed by one or more asterisks (*) beginning in column one. All non-comment lines must end with a semicolon (;).

The mapping table resides in the EYLATMAP member of the DSIPARM data set. Mapping table segments must reside either in the EYLATMAP member or in another member of DSIPARM that is referred to from the mapping table by an %INCLUDE statement.

The general format of the mapping table statement is:

```
►►─from_input─(─loc─┬──────────────────────┬─)─to_name─,─function_name──────►
                    └─,offset─┬─────────┬──┘
                              └─,length─┘

►─┬───────────────────┬─;─────────────────────────────────────────────────►◄
  └─,parm,...,parm────┘
```

where:

*from_input*
: Identifies the type, location, and length of the input data field that is mapped to an alias name or s-word. The type is dependent on the input specified. If the input is MSUSEG, the type is either MSUSEG or HIER. If the input is MSG, the type is either a **keyword=field** within the message or the NetView REXX function MSGSTR. Examples of the MSG input are:

- Using **keyword=field** if input is **MSG**:

  ```
  MAJ(1,1) DESCRIPTION_TEXT,TEXT,DECODE;
  ```

  would get the MAJ= code point value.

  BNJ146I 09/16 11:00 R BKID=FE2 ACT=55 MIN=04 **MAJ=03** ...

- Using **MSGSTR** if input is **MSG**:

  ```
  MSGSTR(2,1) TIME_OCCURRED;
  ```

  would get the second word of the message string.

BNJ146I 09/16 **11:00** R BKID=FE2 ACT=55 MIN=04 MAJ=03 ...

If the input is application data, the type is a keyword within the input. For example,

```
MYDATA(1,*) DESCRIPTION_TEXT,TEXT;
```

The type identifier is followed immediately by parentheses containing the following parameters:

*loc* This parameter references a subset of information within the input record, such as a generic alert subvector. The convention used in the NetView MSUSEG function is adopted here. For example, MSUSEG(0000.93) locates the probable-cause subvector on the alert.

*offset* Is the offset from the start of the input data where data mapping is to begin. If the input is an MSU, then the offset is expressed as a decimal that specifies the number of bytes from the start on the MSU. All other offsets are expressed as word counts from the beginning of the input data.

*length* If the input data is an MSU then the length is the number of bytes in decimal starting from the *byte* position that is to be returned. Otherwise, the length is the number of words to be extracted from the input data.

If you omit this parameter, the entire contents of the input data field will be mapped to the alias name and will be concatenated into 132-character lines. To indicate that you want the entire input data field without this concatenation, specify an asterisk (*) as this parameter.

An asterisk used as the length parameter specifies that the entire input data field will be mapped to the alias name or s-word, and the formatting that currently exists in the input will be kept. Use the asterisk when mapping columnar input data fields or for any input that you want to map as a whole without concatenation.

**Note:** For more information on the MSUSEG and HIER functions, refer to *NetView Customization: Writing Command Lists*.

*to_name*
Is the alias name or s-word as defined in a Tivoli Information Management for z/OS alias table or PIDT.

*function_name*
Command procedures used to modify input data must be written as a function and must do the following:

- Accept the *to_name* value as an input parameter to the function
- Return a value (that replaces the current *to_name* value).

The following syntax must be used to specify a command procedure on a mapping statement:

```
►►─function_name(parm─┬──────────────────┬─)──────────────────────────────►◄
                      └─,parm,...,parm───┘
```

*parm*
Gives one of the following optional parameters:

As an example of how the *function_name* parameter works, consider the following mapping segment statement used to process a generic alert routed through the NetView automation table:

```
MSUSEG(0000.01.10,3,3) DATE_OCCURRED,EYLEXCDT('DATE_OCCURRED'X);
```

This statement causes the hexadecimal value for the date to be extracted from subfield 10 of subvector 01 (the date/time subvector) and assigned to the DATE_OCCURRED field for this record. If the hexadecimal data was X'5C051B' then that is the value assigned to DATE_OCCURRED. However, the EYLEXCDT function specified on this statement causes modification of the DATE_OCCURRED value. EYLEXCDT converts and formats the hexadecimal date value to return the value '05/27/92' and assigns it as the new value of DATE_OCCURRED.

**Note:** The EYLEXCDT function is present in the AutoBridge sample data set along with EYLEXCTM, which converts hexadecimal time values.

**TEXT**
Indicates that the *to_name* variable is to be appended to the variable TEXTLIST of the transaction record. These text data fields can consist of large numbers of lines of variable format containing from 1 to 132 characters per line. Consider the following example:

```
MSUSEG(0000.31.30,3) DESCRIPTION_TEXT,TEXT;
```

In this example, the self-defining text message transported in subvector 31 is added to the freeform text in the record. In the following example, the results of decoding the probable-cause code point (transported in subvector 93) and the user-causes code point (transported in subvector 94) are added to the freeform text in the record:

```
MSUSEG(0000.93.01,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(0000.94.01,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
```

**SEARCH**
Indicates that the *to_name* variable is to be added to the search list of any search transaction created for this input record. The SEARCH keyword allows you to define a conditional create or update request. When the IBCREATE or IBUPDATE command is processed, these fields are searched. In the case of the IBCREATE command if no records are returned, a new record is created. In the case of the IBUPDATE command if no records are returned, the update request is deleted. In either case, if one or more records are returned, the record with the highest RECORDID value is updated. Additionally, if the VERIFIER parameter is specified in the process table for an IBUPDATE request, the field values specified by the VERIFIER statement must match exactly to process an update.

*Table 21. Using SEARCH arguments in IBCREATE or IBUPDATE transactions*

| Transaction | Search results | Action |
|---|---|---|
| IBCREATE | No hits | A new record is created |
| IBCREATE | One or more hits | Highest record id is updated |
| IBUPDATE | No hits | AutoBridge transaction deleted |
| IBUPDATE | One or more hits | Highest record id is updated |

**UPDATE**
Appends the *to_name* variable to the update list of the update transaction. This only

occurs during a conditional create or update (IBCREATE/IBUPDATE with SEARCH keywords) that results in an update. If UPDATE is not specified, that field in the record retains its original value.

In the following MSUSEG statement, the product name transported in subvector 10 replaces the value saved in the field mapped by PRODUCT_NAME:

```
MSUSEG('0000.10(1).11(2).06,3') PRODUCT_NAME,UPDATE;
```

In another example:

```
MSUSEG(0000.92,6,2) DESCRIPTION,DECODE,UPDATE,TEXT;
```

the text string resulting from the description code point transported in subvector 92 is appended to the updated record's freeform text.

**DECODE**

Specifies that the code points in the input data are to be translated into text strings. These text strings are read from NetView and user code point tables.

Table 22 on page 97 summarizes this information.

**Code point translation**

NetView receives code points through network management vector transports (NMVTs) and record formatted maintenance statistics (RECFMS) data records from entry point- and service point-attached devices. AutoBridge provides the facilities to translate these code points into text strings so that the target Tivoli Information Management for z/OS record data is clearer and easier to use.

**Generic and nongeneric alert code points**

MSUs and BNJ146I messages may contain alert code points. The following message fragments contain data designated as *CPL* that represent the code point location:

- BNJ146I generic alert message fragment

  ```
  BNJ146I G TYPE=PERM DESC=CPL PC=CPL,CPL,CPL ACTS=CPL,CPL,CPL
  ```

- BNJ146I nongeneric alert message fragment

  ```
  BNJ146I N PRID=3174 MAJ=CPL MIN=CPL ACT=CPL
  ```

- BNJ146I RECFMS message fragment

  ```
  BNJ146I R BKID=048 ACT=CPL MIN=CPL MAJ=CPL
  ```

Here are several MSU fields containing code points:

- MSUSEG subvectors containing code points

  ```
  MSUSEG(0000.92,5,1)    <Subvector 92 contains the Alert
                          Description>
  MSUSEG(0000.93,3,2)    <Subvector 93 contains one or more
                          probable cause codes>
  MSUSEG(0000.94.01,3,2) <Subvector 94, Subfield 01 contains
                          the User Causes codes>
  MSUSEG(0000.94.81,3,2) <Subvector 94, Subfield 81 contains
  MSUSEG(0000.94.81,5,2)  the Recommended Actions codes>
  ```

**Code point tables**

For generic alerts, the name of the table containing the text is based on the subvector or subfield (NMVT data) where the code originated. For RECFMS, the text associated with an ACT code point is retrieved from an IBM table whose name is based on the block ID prefixed with BNJVM. For example, if the BKID=048, the table name is BNJVM048.

For nongeneric alerts, AutoBridge creates a block ID from the BNJ146I N nongeneric alert based on the PRID.

In the case of BNJ146I generic alert, the subvector/subfield is calculated from the keyword. In the case of MSUs, the subvector/subfield is specified as MSUSEG(0000.*SV.SF.offset,length*).

*Table 22. Generic alert BNJ146I message*

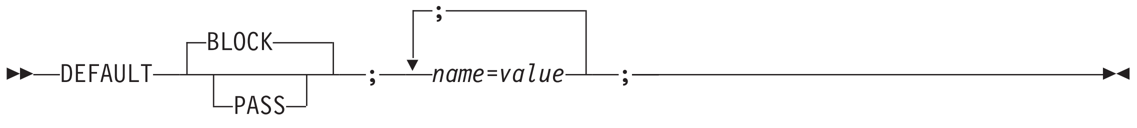
| BNJ146I keyword | MSUSEG SV;SF | Table name | Description |
|---|---|---|---|
| DESC | SV92 | BNJ92TBL/BNJ92UTB | Alert description |
| PC | SV93 | BNJ93TBL/BNJ93UTB | Probable cause |
| ACTS | SV81 | BNJ97TBL/BNJ97UTB | Undetermined cause |
| USER | SV94;SF81 | BNJ94TBL/BNJ94UTB BNJ81TBL/BNJ81UTB | User cause; recommended action |
| FAIL | SV96;SF81 | BNJ96TBL/BNJ96UTB BNJ81TBL/BNJ81UTB | Failure cause; recommended action |
| INST | SV95;SF81 | BNJ95TBL/BNJ95UTB BNJ81TBL/BNJ81UTB | User cause; recommended action |
| ACT | | BNJVM+BKID (BKID may be produced from PSID) | Action code (see note) |
| **Note:** If ACT is specified with the DECODE option, AutoBridge retrieves the BKID or PSID and performs the translation to text. | | | |

## Coding the Filter Table

The filter table provides AutoBridge with the ability to further specify what records are to be processed and when processing is started or stopped.

The filter table must reside in the EYLATFIL member of the DSIPARM data set.

All statements in the filter table should begin on column one of each line (comment lines must have an asterisk (*) in column one). The following is a description of the general format for statements:



where:

**DEFAULT**
Indicates the default option for filtering.

**BLOCK**
Indicates that all input records are blocked and are not processed unless subsequent filter statements cause them to be passed. BLOCK is the default.

**PASS**
Indicates that all inputs are processed unless blocked by subsequent filtering.

*name=value*

Identifies one or more field/value pairs that are checked. *name* is the alias name or s-word of a field in the transaction record. *value* is the actual value that is used in the comparison of the corresponding *to_name* value parsed from the input data.

One or more *name=value* statements can be combined using the AND (&) or the OR (|) operators to perform a comprehensive filtering function. The standard rules of REXX precedence apply. Use parentheses to ensure that sub-expressions are evaluated in the desired order.

A wild card feature is supported that will resolve partial names with a trailing asterisk (*), meaning that any name where the characters precede the asterisk forms a match. For example, "NY1*" means any name starting with the character string 'NY1'.

This is an example of the filter table.

```
***** Filter table ***********************************
**
*
*
DEFAULT BLOCK;

(DOMAIN_ID=CNM01) & (DEVICE_NAME=NY1* | DEVICE_NAME=LA* | DEVICE_NAME=SF*)
& (PRODUCT_SET=PAT);
(EMS_NAME=GTESPAN) & (DEVICE_NAME=VPL* & DESCRIPTION=2*);
(DEVICE_NAME=LAN*) & (LOC_HHMM > 0730 & LOC_HHMM < 1730);
*
*
```

The example is interpreted as follows:

```
If the DOMAIN_ID = CNM01
   and DEVICE_NAME = any value starting with 'NY1'
   or DEVICE_NAME = any value starting with 'LA'
   or DEVICE_NAME = any value starting with 'SF'
   and PRODUCT_SET = PAT
or
if EMS_NAME=GTESPAN
   and DEVICE_NAME = any value starting with 'VPL'
   and DESCRIPTION = any value starting with '2'
or
if DEVICE_NAME = any value starting with 'LAN'
   and LOC_HHMM greater than 07:30
   and LOC_HHMM less than 17:30
then
   process the input record
else
   block the input record from processing
```

## AutoBridge Table Examples

The following sections provide examples to illustrate how you can use AutoBridge to process generic alerts and BNJ146I messages that are trapped by the NetView automation table. Each example shows a NetView automation table statement and the process, mapping, and filter table segments that correspond to it.

## Processing Generic Alerts

This section uses examples to show the relationships between an API invocation statement and the associated AutoBridge table segments for processing generic alerts.

The following example shows a statement coded in the NetView automation table for invoking AutoBridge with parameters for processing a generic alert:

```
IF MSUSEG(0000) ¬= '' THEN
 EXEC (CMD('ABAPI PROCESS_GENALERT BRGDHIPR MSUSEG')
 ROUTE (ONE AUTO1));
```

The call to AutoBridge contains the following parameters:

- PROCESS_GENALERT identifies the name of the process table segment to be used.
- BRGDHIPR is the name of the high priority dispatcher over which the generated transaction will be sent.
- MSUSEG is the AutoBridge keyword that identifies the type of input being processed.

The PROCESS_GENALERT segment of the process table could be coded as follows:

```
******  PROCESS TABLE SEGMENT FOR GENERIC ALERTS *********************
*
BEGIN PROCESS_GENALERT;
PARSE MAPPING=MAP_GENALERT1,FILTER;
ADD_DATA 'OPEN',PROBLEM_STATUS;
ADD_DATA 'NETVIEW',REPORTER_NAME,SEARCH;
ADD_DATA 'D15A',REPORTER_DEPT;
ADD_DATA '3',INITIAL_PRIORITY;
ADD_DATA '555-3454',REPORTER_PHONE;
GLOBALV GETT USER_APPL;
ADD_DATA  USER_APPL,PROGRAM_NAME,SEARCH;
IBCREATE 'INFONETW','PROBLEM';
END PROCESS_GENALERT;
```

For this example, note the following:

- The segment name is the same on both the BEGIN and END statements.
- The PARSE statement identifies the mapping statement to be used for extracting data fields from the alert. It also invokes filtering via the AutoBridge filter table.
- The *from_field* parameter of the ADD_DATA statements (except the last one) contains a literal value. The GLOBALV statement retrieves the task global variable USER_APPL, the value of which is assigned to PROGRAM_NAME in the next ADD_DATA statement.
- The ADD_DATA statements used to add data to the REPORTER_NAME and PROGRAM_NAME fields contain the SEARCH keyword. This makes the IBCREATE function conditional based on the values of these fields. If a record is found that has identical values in these fields, AutoBridge will update this record instead of creating a new record.
- The IBCREATE statement identifies the target database as the Tivoli Information Management for z/OS database and the record type as a PROBLEM record.

The mapping table segment identified on the PARSE statement could be coded as follows:

```
****** THIS IS A MAPPING SEGMENT FOR GENERIC ALERTS *******************
BEGIN  MAP_GENALERT1;
MSUSEG(0000.01.10,3,3) DATE_OCCURRED,EYLEXCDT('DATE_OCCURRED'X);
MSUSEG(0000.01.10,6,3) TIME_OCCURRED,EYLEXCTM('TIME_OCCURRED'X);
MSUSEG(0000.92,5,1) PROBLEM_TYPE,DECODE;
MSUSEG(0000.92,6,2) DESCRIPTION,DECODE;
MSUSEG(0000.93,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(0000.94.01,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(0000.94.81,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(0000.95.01,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(0000.95.81,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(0000.96.01,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(0000.96.81,3,2) DESCRIPTION_TEXT,DECODE,TEXT;
MSUSEG(H1311.81.02,3) NETWORK_NAME;
HIER(4) DEVICE_NAME,EYLEXNAM(DEVICE_NAME),SEARCH;
MSUSEG(0000.31.30,3) DESCRIPTION_TEXT,TEXT;
END MAP_GENALERT1;
```

This mapping table segment has the following features:

- The NetView MSUSEG function is used to parse the generic alert which is routed through the NetView automation table.
- The REXX functions EYLEXCDT and EYLEXCTM are AutoBridge-provided samples for converting hexadecimal date and time values to text and formatting them appropriately.
- DESCRIPTION_TEXT is the alias name of a freeform text field; the actual text is resolved from alert code points (as indicated by the DECODE option) and from subvector 31.

The following example shows a filter table corresponding to the previous examples. Filtering via AutoBridge occurs only if the keyword FILTER is specified on a PARSE statement in the active process table segment.

```
*
*
DEFAULT BLOCK;
NETWORK_NAME=CNM01 | PROBLEM_TYPE=PERM;
*
```

Since the default for this filter table is set to BLOCK, the transaction is processed only if NETWORK_NAME = CNM01 or PROBLEM_TYPE = PERM.

## Processing BNJ146I Messages

This section uses examples to show the relationships between an API invocation statement and the associated process and mapping table segments for processing BNJ146I messages and building transactions to be sent over the NetView Bridge.

The following example shows a statement coded in the NetView automation table for invoking AutoBridge with parameters for processing a BNJ146I message:

```
IF MSGID = 'BNJ146I' & TOKEN(4) = 'G' THEN
 EXEC (CMD('ABAPI PROCESS_BNJ146I BRGDHIPR MSG')
 ROUTE (ONE AUTO1));
```

Here’s an example of a BNJ146I message:

```
09/16 11:00 G TYPE=PERM ALID=17511734 DESC=2000
PSID=USER1 PC=1001,0101 ACTS=1012,1205,3300,0600,3110
TEXT="APPLICATION ABENDED" HIER=RALVS12,CPU,REPTGEN,PROG
```

The PROCESS_BNJ146I segment of the process table could be coded as follows:

```
****  PROCESS TABLE SEGMENT FOR BNJ146I MESSAGE *************
*
BEGIN PROCESS_BNJ146I;
```

```
*TRACE ON;
ADD_DATA 'YES',USE_DATA_VIEW;
ADD_DATA 'OPEN',PROBLEM_STATUS;
ADD_DATA 'NETVIEW',REPORTER_NAME,SEARCH;
ADD_DATA 'D15A',REPORTER_DEPT;
ADD_DATA '3',INITIAL_PRIORITY;
ADD_DATA '555-3454',REPORTER_PHONE;
ADD_DATA 'AUTOBRG',PROGRAM_NAME,SEARCH;
PARSE MAPPING=MAP_BNJ146I;
IBCREATE 'INFONETW','PROBLEM';
END PROCESS_BNJ146I;
```

This segment is similar to the process table segment used for generic alerts in the previous section, except that a different mapping table segment is called from the PARSE statement. The mapping table segment identified on this PARSE statement could be coded as follows:

```
***********  THIS SEGMENT FOR MAPPING BNJ146I MESSAGES *************
BEGIN MAP_BNJ146I;
MSGSTR(1,1) DATE_OCCURRED,EYLEXAYR(DATE_OCCURRED);
MSGSTR(2,1) TIME_OCCURRED;
TYPE(1,1) PROBLEM_TYPE;
DESC(1,1) DESCRIPTION,DECODE;
PC(1,1) DESCRIPTION_TEXT,TEXT,DECODE;
ACTS(1,1) DESCRIPTION_TEXT,TEXT,DECODE;
TEXT(1) DESCRIPTION_TEXT,TEXT;
HIER(1,1) ADDRESS_TEXT,TEXT;
END MAP_BNJ146I;
```

In this mapping table segment, MSGSTR is a NetView function used to extract the date and time from the message buffer. The EYLEXAYR function, supplied with AutoBridge, calculates the current year and adds it to the date value taken from the message, which is in the format *MM/DD*.

Each of the other statements identifies a keyword in the BNJ146I message that is used to locate and extract data from the message buffer.

The statement with the keyword ACTS(1,1) defines the data beginning at offset 1 for a length of one word. Because words are delimited by blanks in REXX, multiple code points will be extracted by this statement and appended to the freeform text field with the alias DESCRIPTION_TEXT.

## Coding the Initialization Table

AutoBridge's initialization values are defined in the NetView DSIPARM data set member EYLATINT. This member, the initialization table, defines the environment in which AutoBridge operates. It is comprised of segments and subsegments of dispatcher, database, and record definitions organized in a nested, hierarchical structure.

The following sections describe and give examples of the structure and syntax of the initialization table. A sample EYLATINT is shipped in SEYLSPL that you can modify to meet your needs.

### Initialization Table Structure

Figure 7 on page 102 illustrates the nested structure of the initialization table.

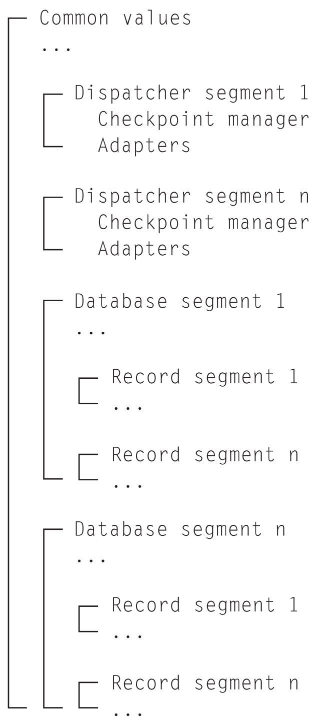


*Figure 7. Initialization table nesting hierarchy.* The initialization table is defined with a strict hierarchical relationship of segments and subsegments.

## Initialization Table Examples

Figure 8 on page 103 and Figure 9 on page 104 show examples of initialization tables on both a resident and remote NetView. These examples define a set of values for the “INFONETW” database only. They also define the bridge dispatchers and database server adapters to be started.

```
*-- EYLATINT ON A RESIDENT NETVIEW -------------------------------------
*-- Common Values ------------------------------------------------------

*-- Type of AutoBridge application, resident or remote -----------------
AUTOBRIDGE = RESIDENT

*-- Date format generated for the checkpoint manager data --------------
DATEFORMAT = U

*-- Wait time (in seconds) for issued commands (EYLSTMGR, MVS D,A) -----
WAITTIME = 29

*-- Remote dispatcher on this NetView ----------------------------------
RDISPATCH = BRGREMOP

*-- Bridge dispatcher, checkpoint mgr and adapters segment -------------
BEGIN DISPATCHER BRGDHIPR
CHECKPT = EYLTCHKH
ADAPTER = INFOBRG1
END DISPATCHER BRGDHIPR

BEGIN DISPATCHER BRGDLOPR
CHECKPT = EYLTCHKL
ADAPTER = INFOBRG2
END DISPATCHER BRGDLOPR

*-- Database segment ---------------------------------------------------

BEGIN INFONETW

DATABASE = INFOMGMT                   type of target database

BRGNETID = NETA                       resident network id
DOMAINID = CNM01                      resident domain id
SEPARATOR = ,                         separation character
*CORRID   = ORIGINAL_PROBLEM          correlation alias name
SEND      = EYLSCIMT                  send transaction processor
RECEIVE   = EYLSCIRC                  receive transaction processor

RETRYNUM = 1                          number of times to resend trans
RETRYINT = 3                          time in minutes between retries

CREPRIV  = MASTER                     (opt) create privilege class
UPDPRIV  = MASTER                     (opt) update privilege class
INQPRIV  = MASTER                     (opt) inquiry privilege class

*-- Record segment -----------------------------------------------------

BEGIN PROBLEM
*VOCAB    = EYLALIA                   problem alias table
IBCREATE = BLGYPRC                    problem create pidt
IBUPDATE = BLGYPRU                    problem update pidt
IBSEARCH = BLGYPRR                    problem retrieve pidt
INQVIEW  = BLGYPRI                    problem inquiry pidt
END PROBLEM

BEGIN CHANGE
VOCAB    = CHGAKA                     change alias table
IBCREATE = BLGYCHC                    change create pidt
IBUPDATE = BLGYCHU                    change update pidt
IBSEARCH = BLGYCHR                    change retrieve pidt
INQVIEW  = BLGYCHI                    change inquiry pidt
END CHANGE

END INFONETW
```

*Figure 8. Example of an initialization table for AutoBridge on a resident NetView*

```
*-- EYLATINT ON A REMOTE NETVIEW ---------------------------------------

*-- Common Values ------------------------------------------------------

*-- Type of AutoBridge Application, Resident or Remote------------------
AUTOBRIDGE = REMOTE

*-- Date format generated for the Checkpoint Manager Data---------------
DATEFORMAT = U

*-- Wait time (in seconds) for issued commands (ie, EYLSTMGR, MVS D,A)--
WAITTIME = 29

*-- Remote Dispatcher on this Remote NetView----------------------------
RDISPATCH = REMOPERA

*-- Bridge Dispatcher, Checkpoint Mgr and Adapters segment--------------
BEGIN DISPATCHER BRGDHIPR
CHECKPT = EYLTCHK1
END DISPATCHER BRGDHIPR

BEGIN DISPATCHER BRGDLOPR
CHECKPT = EYLTCHK2
END DISPATCHER BRGDLOPR

*-- Database Segment----------------------------------------------------

BEGIN INFONETW

DATABASE = INFOMGMT                  type of target database

BRGNETID = NETA                      resident network id
DOMAINID = CNM01                     resident domain id
SEPARATOR = ,                        separation character
*CORRID   = ORIGINAL_PROBLEM         alias name for corrid
SEND      = EYLSCIMT                 send transaction processor
RECEIVE   = EYLSCIRC                 receive transaction processor

RETRYNUM = 2                         number of times to resend trans
RETRYINT = 3                         time in minutes between retries

CREPRIV  = MASTER                    (opt) create privilege class
UPDPRIV  = MASTER                    (opt) update privilege class
INQPRIV  = MASTER                    (opt) inquiry privilege class

*-- Record Segment------------------------------------------------------

BEGIN PROBLEM
VOCAB    = EYLALIA                   problem alias table
IBCREATE = BLGYPRC                   problem create pidt
IBUPDATE = BLGYPRU                   problem update pidt
IBSEARCH = BLGYPRR                   problem retrieve pidt
INQVIEW  = BLGYPRI                   problem inquiry pidt
END PROBLEM

END INFONETW
```

*Figure 9. Example of an initialization table for AutoBridge on a remote NetView*

## Initialization Table Syntax

The following sections describe the syntax for the following data segments in the initialization table:

- Common values
- Dispatcher segment
- Database segment
- Record segment.

The following sections describe the values defined in each of these segments. Each definition is mapped to the corresponding item on the initialization worksheet (see “Initialization Table Worksheet” on page 231 and 232).

## Common Values

This segment defines the NetView Bridge type, the date displayed for checkpoint files, the wait time for issued commands, and the remote dispatcher on the resident NetView.

### Bridge Type

```
                         ┌─RESIDENT─┐
►►──AUTOBRIDGE──=────────┴─REMOTE───┴──────────────────────────────────────────►◄
```

where:

**RESIDENT|REMOTE**
: Specifies the type of NetView Bridge on this host. The target database resides on the resident host. All others are remote. The default value is RESIDENT.

  This value corresponds to item 1 on the initialization table worksheet, which can be found in “NetView AutoBridge Worksheets” on page 231.

### Date Format

```
                       ┌─U─┐
►►──DATEFORMAT──=──────┼─E─┼───────────────────────────────────────────────────►◄
                       └─O─┘
```

where:

**U|E|O**
: Specifies the format of the Created Date and Sent Date values displayed when viewing checkpoint manager files. The default value is U.

| | |
|---|---|
| **U** | USA (mm/dd/yy) |
| **E** | European (dd/mm/yy) |
| **O** | Ordered (yy/mm/dd) |

This value corresponds to item 2 on the initialization table worksheet.

### Wait Time

```
►►──WAITTIME──=──time_in_seconds───────────────────────────────────────────────►◄
```

where:

*time_in_seconds*
Is the time in seconds to wait for the completion of any commands issued by AutoBridge. Any command that has not completed within this time is considered "timed out." An appropriate value depends on the responsiveness of your system. It is recommended that you start with a value of 29 and adjust as necessary.

This value corresponds to item 3 on the initialization table worksheet.

### Remote Dispatcher

```
►►──RDISPATCH──=──remote_dispatcher──────────────────────────────────────────►◄
```

where:

*remote_dispatcher*
Is the name of the remote dispatcher on this host. It is a NetView autotask.

This value corresponds to item 4 on the initialization table worksheet.

## Dispatcher Segment

Bridge dispatchers, checkpoint managers, and bridge adapters on the resident NetView are defined in this segment. The dispatcher name is on the BEGIN/END DISPATCHER segment label.

### Dispatcher Segment Start

```
►►──BEGIN DISPATCHER──bridge_dispatcher──────────────────────────────────────►◄
```

where:

*bridge_dispatcher*
Is the bridge dispatcher being started by this segment.

This statement defines the NetView autotask that is initialized as a dispatcher. Use this statement and the corresponding END statement to delimit segments for each of the bridge dispatchers that will be activated or referenced on the resident host or referenced from the remote host. On both the resident and remote NetView Bridge, the BEGIN DISPATCHER statement associates a bridge dispatcher to a checkpoint manager task.

This value corresponds to items 5, 8, 11, and 14 on the initialization table worksheet.

### Checkpoint Manager

```
►►──CHECKPT──=──checkpoint_manager───────────────────────────────────────────►◄
```

where:

*checkpoint*
Is the name of the checkpoint manager associated with the current dispatcher segment. This NetView subtask is started and stopped by the ABRIDGE command.

This value corresponds to items 6, 9, 12, and 15 on the initialization table worksheet.

### Database Server Adapter

```
►►──ADAPTER──=──adapter──┬──────────────┬──────────────────────────────►◄
                         └─,adapter,...─┘
```

where:

*adapter,adapter,...*
: Names up to four database server adapters associated with the current dispatcher segment. This MVS task will be started or stopped by the ABRIDGE command.

This value corresponds to items 7, 10, 13, and 16 on the initialization table worksheet.

**Note:** The ADAPTER definition is required on the resident AutoBridge only.

### Dispatcher Segment End

```
►►──END DISPATCHER──bridge_dispatcher──────────────────────────────────►◄
```

where:

*bridge_dispatcher*
: Is the same bridge dispatcher identified on the BEGIN statement for this segment.

## Database Segment

This segment contains definitions of database segments and record segments for a specific target database. The database name is on the BEGIN/END *database_segment* segment label.

### Database Segment Start

```
►►──BEGIN──database_segment────────────────────────────────────────────►◄
```

where:

*database_segment*
: Is the database segment that is defined in this segment.

This value corresponds to item 17 on the initialization table worksheet.

### Database Type

```
                 ┌─INFOMGMT──────┐
►►──DATABASE──=──┴─database_name─┴─────────────────────────────────────►◄
```

where:

*database_name*
: Specifies the target database type. AutoBridge supports the Tivoli Information Management for z/OS database (INFOMGMT) only.

This value corresponds to item 18 on the initialization table worksheet.

## Resident Network ID

```
►►──BRGNETID──=──resident_netid──────────────────────────────────────────►◄
```

where:

*resident_netid*
Is the ID of the network where the target Tivoli Information Management for z/OS database resides. If you specify an asterisk **&**, the network ID defaults to the one determined by VTAM®.

This value corresponds to item 19 on the initialization table worksheet.

## Resident Domain ID

```
►►──DOMAINID──=──resident_domainid───────────────────────────────────────►◄
```

where:

*resident_domainid*
Is the NetView domain ID where the NetView Bridge adapter is active.

This value corresponds to item 20 on the initialization table worksheet.

## Separator Character

```
►►──SEPARATOR──=──separator_character────────────────────────────────────►◄
```

where:

*separator_character*
Is the control parameter of the character used to separate list items in the Tivoli Information Management for z/OS data.

This value corresponds to item 21 on the initialization table worksheet.

## Correlation ID Alias Name

```
►►──CORRID──=──alias_name────────────────────────────────────────────────►◄
```

where:

*alias_name*
Is the alias name where the correlation ID generated by AutoBridge will be saved in the Tivoli Information Management for z/OS record. You can omit this field if no correlation ID is to be saved in the record.

This value corresponds to item 22 on the initialization table worksheet.

## Send Command Name

```
►►──SEND──=──send_transaction──────────────────────────────────────►◄
```

where:

*send_transaction*
Is the name of the command to invoke when sending a transaction request to the target database. AutoBridge supplies EYLSCIMT as the interface to Tivoli Information Management for z/OS.

This value corresponds to item 23 on the initialization table worksheet.

## Receive Command Name

```
►►──RECEIVE──=──receive_transaction────────────────────────────────►◄
```

where:

*receive_transaction*
Is the name of the command to invoke when receiving a transaction reply from the target database. AutoBridge supports EYLSCIRC.

This value corresponds to item 24 on the initialization table worksheet.

## Number of Retries

```
►►──RETRYNUM──=──resend_times──────────────────────────────────────►◄
```

where:

*resend_times*
Is the number of times to attempt to send a transaction. This value may be between 1–9.

This value corresponds to item 25 on the initialization table worksheet.

## Retry Interval

```
►►──RETRYINT──=──resend_interval───────────────────────────────────►◄
```

where:

*resend_interval*
Is the number of whole minutes between retry attempts.

This value corresponds to item 26 on the initialization table worksheet.

## Create Privilege Class

```
►►──CREPRIV──=──create_privilege_class──────────────────────────────────►◄
```

where:

*create_privilege_class*
Is the privilege class used by the Tivoli Information Management for z/OS API when performing a create. If omitted, the transaction uses the initialization privilege class defined in INFOBRDS.

This value corresponds to item 27 on the initialization table worksheet.

## Update Privilege Class

```
►►──UPDPRIV──=──update_privilege_class──────────────────────────────────►◄
```

where:

*update_privilege_class*
Is the privilege class used by the Tivoli Information Management for z/OS API when performing an update. If omitted, the transaction uses the initialization privilege class defined in INFOBRDS.

This value corresponds to item 28 on the initialization table worksheet.

## Inquiry Privilege Class

```
►►──INQPRIV──=──inquiry_privilege_class─────────────────────────────────►◄
```

where:

*inquiry_privilege_class*
Is the privilege class used by the Tivoli Information Management for z/OS API when performing an inquiry. If omitted, the transaction uses the initialization privilege class defined in INFOBRDS.

This value corresponds to item 29 on the initialization table worksheet.

## Database Segment End

```
►►──END──database_segment───────────────────────────────────────────────►◄
```

where:

*database_segment*
Is the same database segment identified on the corresponding BEGIN statement for this segment. Place the END statement for the database segment after all associated record segments within the database segment. See Figure 7 on page 102.

## Record Segment

This segment groups together a series of values that apply to one type of database record. The record type is on the BEGIN/END *record_type* segment label. Make a segment for each unique type of record you want to process.

### Record Segment Start

```
►►──BEGIN──record_type──────────────────────────────────────────────►◄
```

where:

*record_type*
: Identifies the type of record defined in this segment.

This value corresponds to items 30, 36, 42, and 48 on the initialization table worksheet.

### Alias Table

```
►►──VOCAB──=──alias_table───────────────────────────────────────────►◄
```

where:

*alias_table*
: Is the name of the alias table that is used for this transaction.

  This value corresponds to items 31, 37, 43, and 49 on the initialization table worksheet.

### Create PIDT

```
►►──IBCREATE──=──create_PIDT────────────────────────────────────────►◄
```

where:

*create_PIDT*
: Is the name of the PIDT or Data View Record used for the IBCREATE transaction.

  This value corresponds to items 32, 38, 44, and 50 on the initialization table worksheet.

### Update PIDT

```
►►──IBUPDATE──=──update_PIDT────────────────────────────────────────►◄
```

where:

*update_PIDT*
: Is the name of the PIDT or Data View Record used for the IBUPDATE transaction.

  This value corresponds to items 33, 39, 45, and 51 on the initialization table worksheet.

## Search PIDT

```
►►──IBSEARCH──=──retrieve_PIDT──────────────────────────────────────────►◄
```

where:

*search_PIDT*
Is the name of the PIDT or Data View Record used for the IBSEARCH transaction.

This value corresponds to items 34, 40, 46, and 52 on the initialization table worksheet.

## Inquiry PIDT

```
►►──INQVIEW──=──inquiry_PIDT────────────────────────────────────────────►◄
```

where:

*inquiry_PIDT*
Specifies the name of the PIDT or Data View Record used for the IBCREATE or the IBSEARCH transactions.

This value corresponds to items 35, 41, 47, and 53 on the initialization table worksheet.

## Record Segment End

```
►►──END──record_type────────────────────────────────────────────────────►◄
```

where:

*record_type*
Is the same record type identified on the corresponding BEGIN statement for this segment.

# 9

# NetView AutoBridge Commands

AutoBridge provides commands that let you:

- Invoke AutoBridge from the NetView automation table
- Submit data directly to Tivoli Information Management for z/OS. from other applications
- Start, stop, or recycle AutoBridge and AutoBridge tasks
- Manage AutoBridge checkpoint files
- Manage AutoBridge's process, mapping, and filter tables
- Turn tracing on or off for the AutoBridge application.

To simplify the task of issuing AutoBridge commands, AutoBridge provides an operator interface—a set of full-screen panels—for specifying command parameters. You can access the panels by entering the appropriate command or by selecting them from the AutoBridge main menu panel.

Table 23 lists the AutoBridge commands along with their respective actions and synonyms.

*Table 23. Summary of AutoBridge commands*

| Command | Synonym | Where used | Action | Described in |
|---|---|---|---|---|
| EYLEAPI | ABAPI | NetView automation table, other application, or command processor | Invokes AutoBridge with parameters (process segment name, bridge dispatcher name, and input data) | "Invoking AutoBridge" on page 114 |
| EYLSCSUB | ABSUB | Other application or command processor, any NetView Network Communication Control Facility (NCCF) panel | Submits, requests, or deletes data to or from AutoBridge's checkpoint managers | "Handling Checkpoint Manager Transactions" on page 116 |
| EYLEHBRG | ABRIDGE | NetView automation table or any NetView NCCF panel | Starts, stops, or recycles the AutoBridge API and AutoBridge dispatcher, checkpoint, and adapter tasks | "Starting/Stopping/Recycling AutoBridge and Its Components" on page 119 |
| EYLEUMEN | ABMENU | Any NetView NCCF panel | Displays the AutoBridge main menu | "Using the AutoBridge Main Menu" on page 120 |
| EYLEUSRS | ABSRS | AutoBridge main menu, any NetView NCCF panel | Displays a panel for starting, recycling, and stopping AutoBridge | "Starting/Recycling/Stopping the NetView Bridge Dispatchers or Adapters" on page 121 |
| EYLEUCKP | ABCHECKP | AutoBridge main menu, any NetView NCCF panel | Displays a panel for managing the AutoBridge checkpoint files | "Managing Checkpoint Transactions" on page 121 |

Table 23. Summary of AutoBridge commands (continued)

| Command | Synonym | Where used | Action | Described in |
|---|---|---|---|---|
| EYLEUFMP | ABTABLES | AutoBridge main menu, any NetView NCCF panel | Displays a panel for managing the filter, mapping, and process tables | “Managing the AutoBridge Tables” on page 125 |
| EYLEUTRC | ABTRACE | AutoBridge main menu, any NetView NCCF panel | Displays a panel for setting tracing on or off for AutoBridge | “Setting AutoBridge Tracing On or Off” on page 126 |

## Invoking AutoBridge

The command used to invoke AutoBridge is ABAPI (a synonym for EYLEAPI). Entries in the NetView automation table may issue this command to request an AutoBridge transaction based on an appropriate alert or message. The command can also be issued by applications and command processors that are written in any high-level programming language.

The ABAPI command uses the following syntax:

```
                                                    ┌─MSUSEG─┐
►►──ABAPI──process_segment_name──bridge_dispatcher_name──┼─MSG────┼──────────────►◄
                                                    └─input──┘
```

where:

*process_segment_name*
: Specifies the process table (EYLATPRO) segment that contains the process steps to perform.

*bridge_dispatcher_name*
: Specifies the bridge dispatcher on the resident NetView that will process this transaction. The record will be processed by the checkpoint task associated with this dispatcher.

**MSUSEG**
: Specifies that an MSU has driven this request either directly from the NetView automation table or through an intermediate application.

**MSG**
: Specifies that a message has driven this request directly from the NetView automation table.

*input*
: Specifies the name of a *task global variable* containing input data.

Task global variables are REXX variables that have been declared with the NetView GLOBALV PUTT command. Such variables may be defined in two forms—*simple* and *stemmed* (compound). A simple variable is defined as follows:

```
TECHNICIAN = 'JOE'
DEVICE_NAME = 'CNM0199LU'
DESCRIPTION = 'REPORT OF A NETWORK FAILURE'
```

Input for AutoBridge is stored in simple variables in a *keyword* = *data* format as shown for the variable “buffer” in the example on page 115.

Stemmed variables are specified as a series of *stem.tail* values. The tail value of each variable in a series is incremented over the previous one, with the first tail value set to zero. The zero-tailed variable must be set to the number of stemmed variables to be accessed in the series. The variables will be accessed starting at the first tail until the value in stem.0 is met.

These are examples of stemmed variables:

```
SMALL_VAR.0 = 1
SMALL_VAR.1 = 'THIS IS A SMALL STEMMED VARIABLE'

MEDIUM_VAR.0 = 3
MEDIUM_VAR.1 = 'THIS IS A SLIGHTLY BIGGER STEMMED'
MEDIUM_VAR.2 = 'VARIABLE CONTAINING SEVERAL LINES'
MEDIUM_VAR.3 = 'OF TEXT'

BIG_VAR.0 = 7
BIG_VAR.1 = 'AND THIS IS OUR BIGGEST EXAMPLE OF'
BIG_VAR.2 = 'A STEMMED VARIABLE.  EACH LINE OF '
BIG_VAR.3 = 'THIS IS SAVED AS A SEPARATE LINE  '
BIG_VAR.4 = 'IN THE FREEFORM TEXT AREA OF THE  '
BIG_VAR.5 = 'DATABASE.  ASSUMING OF COURSE THAT'
BIG_VAR.6 = 'YOU SET UP THE ALIAS NAME PROPERLY'
BIG_VAR.7 = 'TO MAKE THAT HAPPEN.              '
BIG_VAR.8 = 'THESE LAST TWO STEMS ARE NOT      '
BIG_VAR.9 = 'ACCESSED BY THE AUTOBRIDGE API.   '
```

**Notes:**

1. In the previous example, BIG_VAR.8 and BIG_VAR.9 will not be accessed because they exceed the value set in BIG_VAR.0.
2. Stemmed variables that are used to contain data that is parsed in the mapping table must contain input data that is in a “keyword = data” format, as illustrated by the following example:

```
MAPPED_TEXT.0 = 5
MAPPED_TEXT.1 = 'MAPTEXT=THIS DATA WILL BE PARSED    '
MAPPED_TEXT.2 = 'IN THE MAPPING TABLE.  THE KEYWORD  '
MAPPED_TEXT.3 = 'IN THIS INPUT SHOULD BE IN THE      '
MAPPED_TEXT.4 = 'FIRST LINE OF THE DATA (THE VALUE   '
MAPPED_TEXT.5 = 'OF THE VARIABLE WITH THE TAIL OF 1).'
```

These first two examples are examples of calls to AutoBridge from the NetView automation table. The third example shows how AutoBridge can be requested by another application.

- In this example, an Alert (MS Major Vector X'0000') with a second subvector 10 (Product Set ID) of ’IBM LAN MANAGER’ drives the API. (See *NetView Customization: Writing Command Lists* for more information on the MSUSEG function.)

```
IF MSUSEG(0000.10.(2)) = . 'IBM LAN MANAGER' . THEN
  EXEC (CMD('ABAPI LANPROC BRGDHIPR MSUSEG') ROUTE(ONE BRIDGE));
```

- In this example, an alert message (BNJ146I) containing RECFMS data drives AutoBridge.

```
IF MSGID = 'BNJ146I' & TOKEN(4) = 'R' & TEXT = MESSAGE
 THEN EXEC(CMD('ABAPI SNAPROC BRGDLOPR MSG') ROUTE (ONE BRIDGE));
```

- In this example, an application program invokes AutoBridge to pass data to the target database.

```
buffer = 'device_name = ny13820 fftext = This is control data for
this particular device that will be sent to Tivoli Information Management for z/OS.'
```

```
'GLOBALV PUTT buffer'

If new_information = 'YES' Then
  'ABAPI AUTOPROC BRGAHIPR buffer'
```

## Handling Checkpoint Manager Transactions

The ABSUB (EYLSCSUB) command enables you to directly submit, retrieve, update, or delete data to or from AutoBridge's checkpoint manager, bypassing the AutoBridge API. This command has five function keywords that you can use to either PUT, GET, LIST, UPDATE, or DELETE AutoBridge transactions in a checkpoint file as follows:

**PUT** The ABSUB PUT function saves transaction data in a checkpoint file and submits it to the database. Use this function to submit Tivoli Information Management for z/OS transactions that are built by other applications or command processors. Because the ABSUB command bypasses the AutoBridge API, it lets you submit transactions to Tivoli Information Management for z/OS. without having to pass them through AutoBridge's process and mapping tables. This way, your automation programs can take advantage of AutoBridge's recovery and post-processing abilities without your having to re-define their output in the AutoBridge tables.

The checkpoint manager task responds to this command by generating a correlation ID for the transaction, which it returns in the variable EYLCORID. Programs that submit transactions in this way can use this value to query transaction status using the ABSUB GET command or delete transactions with the ABSUB DELETE command.

**GET** The ABSUB GET command retrieves a specific record from a checkpoint file and returns it in the form of a multiline message. The multi-line message begins with message EYL551I, which is followed by EYL553I messages that each contain a piece of transaction data.

**LIST** The ABSUB LIST command returns control information from a specific checkpoint file or from all checkpoint files. This control information is returned in a multiline message that begins with message EYL561I, followed by EYL563I messages.

**UPDATE** The ABSUB UPDATE command updates a transaction in the checkpoint file and submits it to the NetView Bridge API. This command allows transactions in the checkpoint file to be corrected and resubmitted.

**DELETE** The ABSUB DELETE command deletes a transaction from the checkpoint file where it resides and returns a confirmation message, EYL547I.

**Note:** All parameters for the specified PUT, GET, UPDATE, or DELETE function are required in the ABSUB command. For the LIST function, you must specify a dispatcher ID or the ALL keyword. If you do not specify the required parameters for these functions, an error message is issued and command processing is halted.

The ABSUB command has the following syntax:

```
►►──ABSUB──┬─PUT──dispatcher──database──rec_type──transid──parmvar──────────┬──────►◄
           ├─GET──dispatcher──corrid────────────────────────────────────────┤
           ├─LIST──┬─dispatcher─┬───────────────────────────────────────────┤
           │       └─ALL────────┘                                           │
           ├─UPDATE──corrid──dispatcher──database──rec_type──transid──parmvar─┤
           └─DELETE──dispatcher──corrid─────────────────────────────────────┘
```

where:

*dispatcher*
: Specifies the ID of the dispatcher used to process this transaction. This value determines the checkpoint manager to which the transaction output is routed.

*database*
: Specifies the name of the database as defined in the initialization table.

*rec_type*
: Specifies the record type for the transaction as defined in the initialization table.

*transid*
: Specifies the type of transaction: IBCREATE, IBSEARCH, or IBUPDATE.

*parmvar*
: Specifies the name of a variable containing a list of task global variables required to process the transaction. These variables can be in any of the following three forms:

**Alias names** The name of the variable is the same as the alias name of the field that will hold this variable's contents.

**S-words** The name of the variable is the same as the structured word index of the field that will hold this variable's contents.

**List names** One of the following variable names that contains a list of variables used by the checkpoint manager task:

**VERIFIER** One or more field/value pairs that must be checked against the matching database record as a means of verification prior to updating the record. See "VERIFIER Function Syntax" on page 87 for details on VERIFIER.

**Note:** If VERIFIER is present, it must be placed before TEXTLIST, SEARCHLIST, and UPDATELIST; otherwise, RESPCODE 2 REASONCODE 24 (RC2 RC24 INVALID VERIFIER) will be returned.

**TEXTLIST** The list of stemmed variables containing freeform text data.

**SEARCHLIST**
The list of variables specifying which fields are to be included in a search argument when processing a conditional IBCREATE or IBSEARCH transaction.

**UPDATELIST**
The list of variables specifying which fields are to replace the original data when updating a record. If

UPDATELIST includes a freeform text s-word or alias, RESPCODE 12 REASONCODE 32 (RC12 RC32 DYNAMIC ALLOCATION ERROR) will be received. Put text s-words or aliases in UPDATETEXTLIST and then include UPDATETEXTLIST in UPDATELIST. See the following example for the proper format.

To differentiate these list names variables from the alias name, the keyword *EYLLIST* is always the first value in the list.

*corrid*
Specifies the correlation ID assigned to the transaction you are attempting to GET, UPDATE, or DELETE.

**ALL**
Specifies that transactions will be listed for all checkpoint managers.

The following is an example of the ABSUB command used to submit a create transaction to the checkpoint manager associated with the NetView Bridge dispatcher BRGHIPR.

```
'ABSUB PUT BRGDHIPR INFONETW PROBLEM IBCREATE PARMVAR'
```

where:

```
PARMVAR = DATE_OCCURRED
          TIME_OCCURRED
          USE_DATA_VIEW
          DOMAINID
          DEVICE_NAME
          PROBLEM_TYPE
          INITIAL_PRIORITY
          DESCRIPTION
          VERIFIER
          TEXTLIST
          SEARCHLIST
          UPDATELIST
```

and where:

```
DATE_OCCURRED    =   02/28/1998
TIME_OCCURRED    =   12:57
USE_DATA_VIEW    =   YES
DOMAINID         =   CNM01
DEVICE_NAME      =   CNM0199LU
PROBLEM_TYPE     =   HARDWARE
INITIAL_PRIORITY =   3
DESCRIPTION      =   'TOKEN-RING INOPERATIVE'

VERIFIER         = EYLLIST REPORTER_NAME INITIAL_PRIORITY
TEXTLIST         = EYLLIST DESCRIPTION_TEXT STATUS_TEXT
SEARCHLIST       = EYLLIST DEVICE_NAME DOMAINID
UPDATELIST       = EYLLIST DATE_OCCURRED TIME_OCCURRED
                           PROBLEM_TYPE UPDATETEXTLIST

DESCRIPTION_TEXT.0 = 5
DESCRIPTION_TEXT.1 = 'TOKEN-RING INOPERATIVE'
DESCRIPTION_TEXT.2 = 'TOKEN-RING FAULT DOMAIN'
DESCRIPTION_TEXT.3 = 'I120 - REVIEW LINK DETAILED DATA'
DESCRIPTION_TEXT.4 = 'IF PROBLEM PERSISTS THEN DO THE FOLLOWING'
DESCRIPTION_TEXT.5 = 'CONTACT TOKEN-RING ADMINISTRATOR RESPONSIBLE'

STATUS_TEXT.0 = 2
STATUS_TEXT.1 = 'THIS ERROR DETECTED VIA MSU IN NETVIEW AND'
STATUS_TEXT.2 = 'AUTOMATICALLY OPENED VIA AUTOBRIDGE'
```

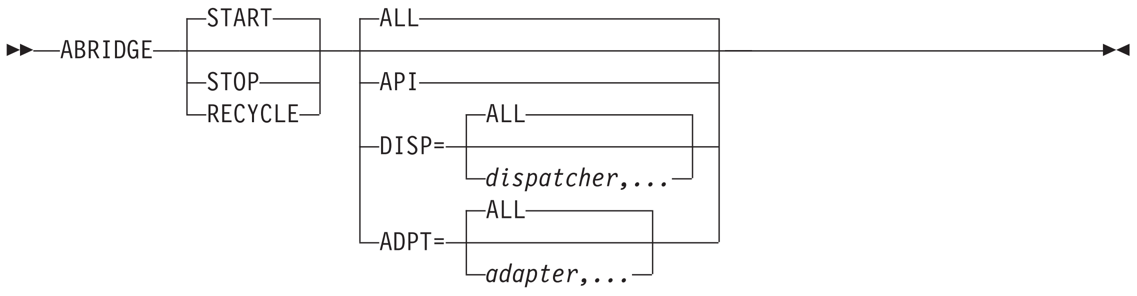
```
UPDATETEXTLIST = EYLLIST STATUS_TEXT

REPORTER_NAME = ABRIDGE
INITIAL_PRIORITY = 1
```

## Starting/Stopping/Recycling AutoBridge and Its Components

The ABRIDGE (or EYLEHBRG) command allows you to start, stop, or recycle AutoBridge and its separate components. It can be invoked as follows:

- From the NetView automation table during NetView startup or shutdown
- By an application used to start, stop, or recycle AutoBridge
- By a user on the command line.

It has the following syntax:



where:

**START**
Initializes AutoBridge or specific AutoBridge components. START is the default parameter for the ABRIDGE command.

**STOP**
Stops the processing of AutoBridge or specific AutoBridge components.

**RECYCLE**
Conditionally stops and then restarts AutoBridge or specific AutoBridge components.

**ALL**
Specifies that all AutoBridge components will be started, stopped, or recycled. ALL is the default parameter for the ABRIDGE command.

**API**
Specifies that the AutoBridge API will be started, stopped, or recycled.

**DISP**
Specifies that the AutoBridge dispatchers will be started, stopped, or recycled as specified by the following parameters:

**ALL**
Specifies that all bridge dispatchers defined in the initialization table will be started, stopped, or recycled.

*dispatcher,...*
Specifies which of the bridge dispatchers to start, stop, or recycle. You can specify dispatchers that are not listed in the initialization table.

**ADPT**
Specifies that the AutoBridge adapters will be started, stopped, or recycled as specified by the following parameters:

**ALL**
Specifies that all bridge adapters defined in the initialization table will be started, stopped, or recycled.

*adapter,...*
Specifies which bridge adapters to start, stop, or recycle. You can specify adapters that are not listed in the initialization table.

As an example of the syntax of the ABRIDGE command, the following command initializes all AutoBridge components:

```
ABRIDGE START ALL
```

Any AutoBridge components already active are not restarted, but remain active.

The following command stops all bridge dispatchers defined in the initialization table:

```
ABRIDGE STOP DISP=ALL
```

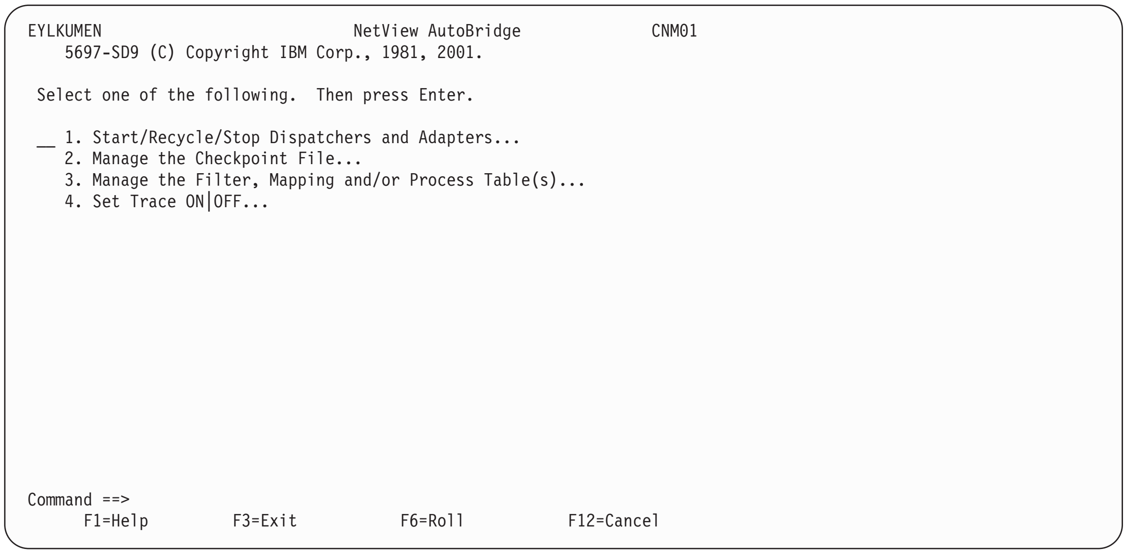
The following command stops (if active) and starts the bridge adapters: BRGAHIPR and BRGAHIP2.

```
ABRIDGE RECYCLE ADPT=BRGAHIPR,BRGAHIP2
```

These adapters do not have to be defined in the initialization table.

## Using the AutoBridge Main Menu

The AutoBridge main menu provides access to all of the AutoBridge panels. To access the AutoBridge main menu, enter **ABMENU** (or **EYLEUMEN**) from any NetView NCCF panel. This is the main menu panel.



```
EYLKUMEN                           NetView AutoBridge             CNM01
    5697-SD9 (C) Copyright IBM Corp., 1981, 2001.

 Select one of the following.  Then press Enter.

 __ 1. Start/Recycle/Stop Dispatchers and Adapters...
    2. Manage the Checkpoint File...
    3. Manage the Filter, Mapping and/or Process Table(s)...
    4. Set Trace ON|OFF...










Command ==>
      F1=Help        F3=Exit          F6=Roll          F12=Cancel
```

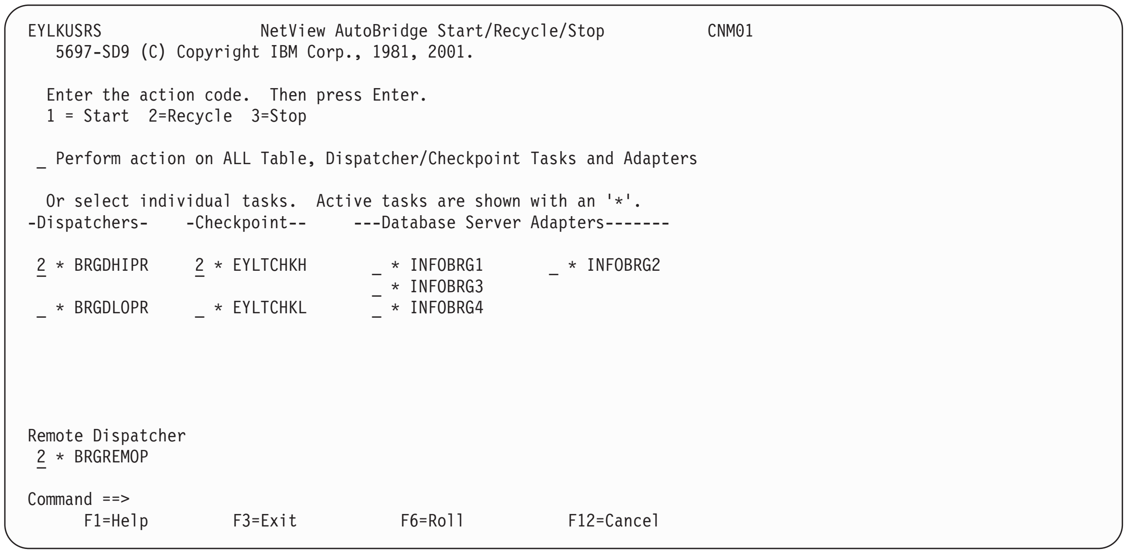
There are three ways to select an option from the main menu:
- Type the number of the option you want to select and press Enter.
- Place the cursor next to the option you want to select and press Enter.
- Type the corresponding command on the command line and press Enter.

You can also use the command line to enter NetView commands.

## Starting/Recycling/Stopping the NetView Bridge Dispatchers or Adapters

The ABSRS command (EYLEUSRS), option 1 on the AutoBridge main menu, enables you to start, stop, or recycle one or more NetView Bridge dispatcher, checkpoint, and adapter tasks. You can start these tasks separately; however, for NetView Bridge transactions to perform successfully on a given dispatcher, all tasks associated with it must be active.

This is the Tivoli Information Management for z/OS NetView AutoBridge Start/Recycle/Stop panel.



```
EYLKUSRS                 NetView AutoBridge Start/Recycle/Stop           CNM01
   5697-SD9 (C) Copyright IBM Corp., 1981, 2001.

  Enter the action code.  Then press Enter.
  1 = Start  2=Recycle  3=Stop

 _ Perform action on ALL Table, Dispatcher/Checkpoint Tasks and Adapters

  Or select individual tasks.  Active tasks are shown with an '*'.
-Dispatchers-    -Checkpoint--     ---Database Server Adapters-------

 2 * BRGDHIPR     2 * EYLTCHKH       _ * INFOBRG1       _ * INFOBRG2
                                     _ * INFOBRG3
 _ * BRGDLOPR     _ * EYLTCHKL       _ * INFOBRG4





Remote Dispatcher
 2 * BRGREMOP

Command ==>
      F1=Help        F3=Exit          F6=Roll          F12=Cancel
```

This panel gives you the option of performing actions on all component tasks or on individual ones. On this panel, a dispatcher, its associated checkpoint task, and up to four associated database server adapters are grouped together on two lines. Active tasks are shown with an asterisk (*). Note how a dispatcher and its associated tasks are arranged on this panel. The high-priority dispatcher BRGDHIPR is grouped together with the EYLTCHKH checkpoint task and database server adapters INFOBRG1, INFOBRG2, and INFOBRG3.

To start, recycle (stop and immediately restart), or stop all component tasks, enter an action code in the first field on the panel.

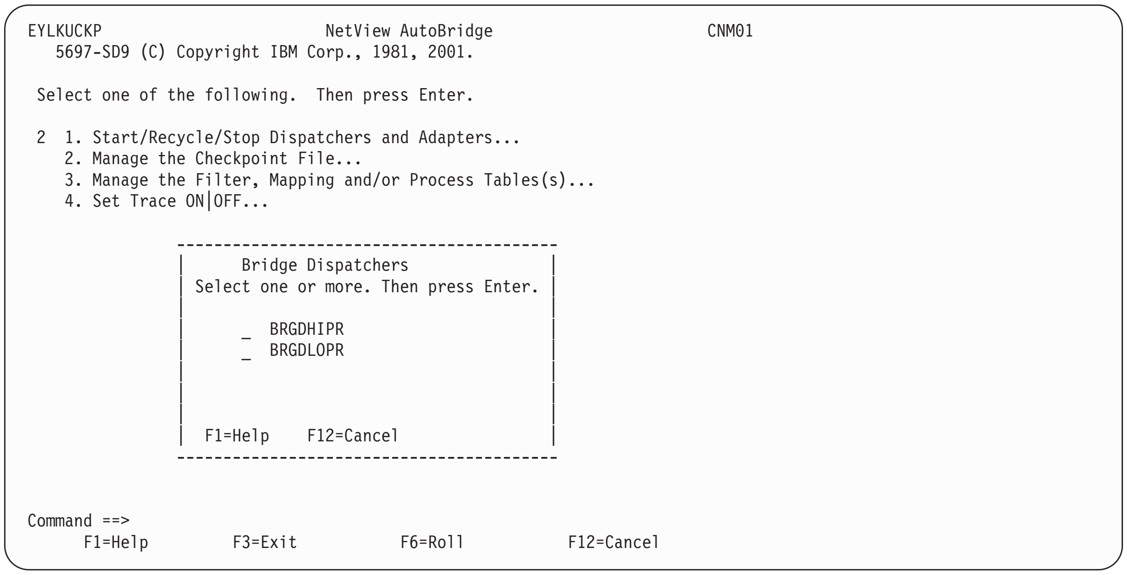
To perform an action on a specific bridge dispatcher, checkpoint task, database server adapter, or the remote dispatcher, tab to the appropriate task and enter the action code beside it.

The example shows action codes typed in to recycle the high-priority bridge dispatcher, its associated checkpoint task, and the remote dispatcher.

## Managing Checkpoint Transactions

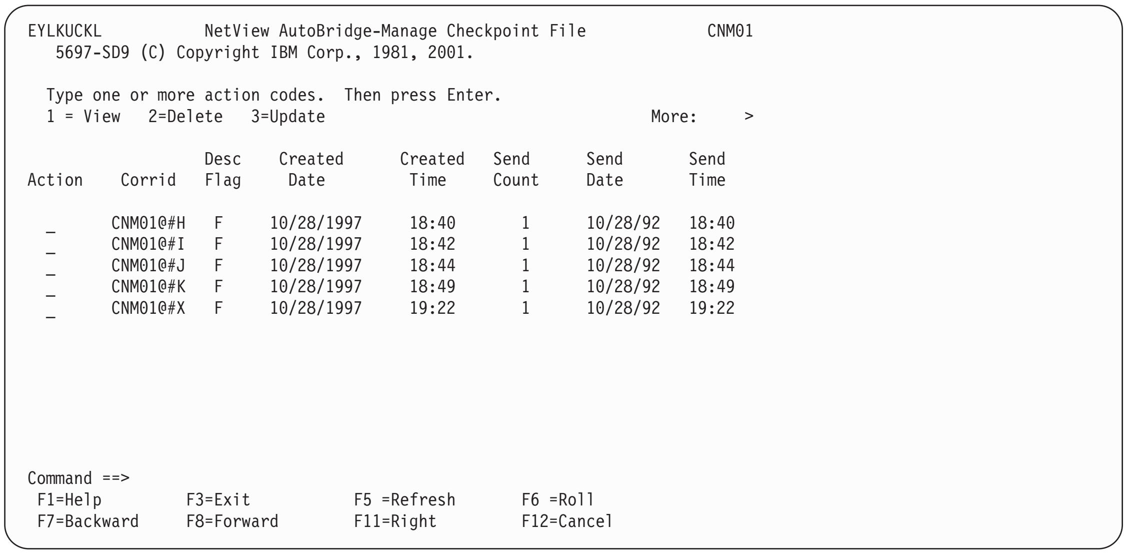
The ABCHECKP (EYLKUCKP) command, option 2 on the AutoBridge main menu, enables you to manage AutoBridge transaction records stored in the checkpoint file.

A pop-up panel appears as shown in this panel if there is more than one bridge dispatcher on your system. It allows you to select one or more dispatchers to manage by typing a slash (/) next to them and pressing Enter. If your system has only one dispatcher, the pop-up panel does not appear.



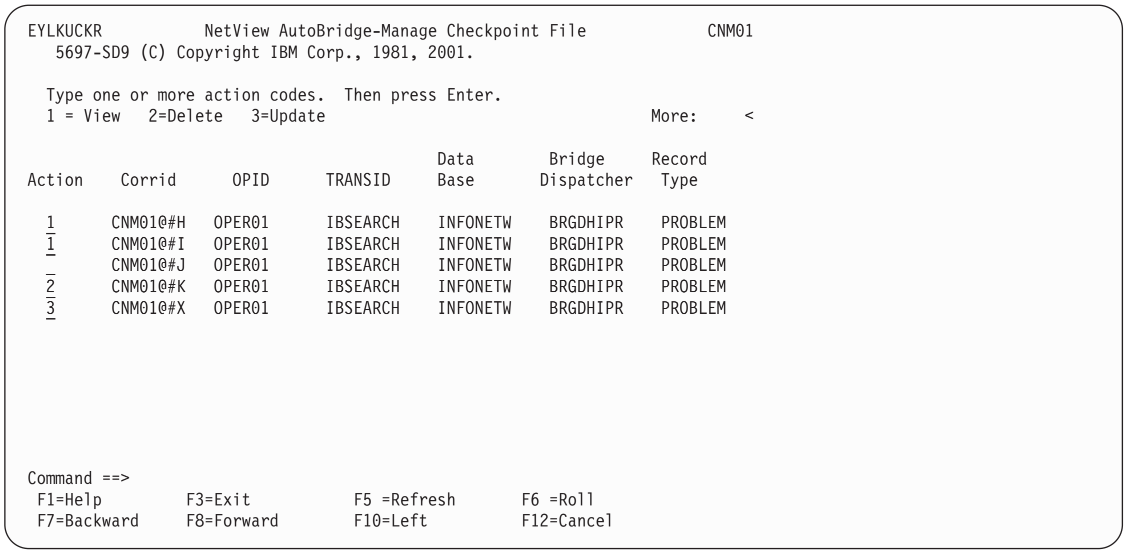
```
EYLKUCKP                        NetView AutoBridge                       CNM01
   5697-SD9 (C) Copyright IBM Corp., 1981, 2001.

 Select one of the following.  Then press Enter.

 2  1. Start/Recycle/Stop Dispatchers and Adapters...
    2. Manage the Checkpoint File...
    3. Manage the Filter, Mapping and/or Process Tables(s)...
    4. Set Trace ON|OFF...

                ----------------------------------------
                |      Bridge Dispatchers              |
                | Select one or more. Then press Enter. |
                |                                      |
                |      _  BRGDHIPR                     |
                |      _  BRGDLOPR                     |
                |                                      |
                |                                      |
                |                                      |
                |  F1=Help    F12=Cancel               |
                ----------------------------------------


Command ==>
      F1=Help         F3=Exit          F6=Roll          F12=Cancel
```

After you have entered your selection, the Manage the Checkpoint File panel appears as shown in this panel.



```
EYLKUCKL           NetView AutoBridge-Manage Checkpoint File            CNM01
   5697-SD9 (C) Copyright IBM Corp., 1981, 2001.

  Type one or more action codes.  Then press Enter.
  1 = View   2=Delete   3=Update                                  More:     >

                   Desc    Created      Created   Send      Send       Send
Action    Corrid   Flag     Date         Time     Count     Date       Time

  _      CNM01@#H   F     10/28/1997     18:40       1      10/28/92   18:40
  _      CNM01@#I   F     10/28/1997     18:42       1      10/28/92   18:42
  _      CNM01@#J   F     10/28/1997     18:44       1      10/28/92   18:44
  _      CNM01@#K   F     10/28/1997     18:49       1      10/28/92   18:49
  _      CNM01@#X   F     10/28/1997     19:22       1      10/28/92   19:22







Command ==>
 F1=Help        F3=Exit          F5 =Refresh       F6 =Roll
 F7=Backward     F8=Forward       F11=Right         F12=Cancel
```

This panel shows control information for each transaction record in the checkpoint file associated with the selected dispatchers. Press PF11 to shift right and view the rest of the line. These are the fields that are displayed after PF11 has been pressed to shift the display to the right.



```
EYLKUCKR          NetView AutoBridge-Manage Checkpoint File            CNM01
   5697-SD9 (C) Copyright IBM Corp., 1981, 2001.

  Type one or more action codes.  Then press Enter.
  1 = View   2=Delete   3=Update                               More:     <

                                        Data        Bridge     Record
Action    Corrid      OPID      TRANSID     Base       Dispatcher   Type

  1      CNM01@#H   OPER01      IBSEARCH    INFONETW    BRGDHIPR    PROBLEM
  1      CNM01@#I   OPER01      IBSEARCH    INFONETW    BRGDHIPR    PROBLEM
  _      CNM01@#J   OPER01      IBSEARCH    INFONETW    BRGDHIPR    PROBLEM
  2      CNM01@#K   OPER01      IBSEARCH    INFONETW    BRGDHIPR    PROBLEM
  3      CNM01@#X   OPER01      IBSEARCH    INFONETW    BRGDHIPR    PROBLEM








Command ==>
 F1=Help         F3=Exit           F5 =Refresh       F6 =Roll
 F7=Backward     F8=Forward        F10=Left          F12=Cancel
```

To view, update, or delete a transaction in the checkpoint file, tab to the transaction and enter one of the following action codes beside it:

| | |
|---|---|
| **1** | To view the checkpoint file transaction |
| **2** | To delete the checkpoint file transaction |
| **3** | To update the checkpoint file transaction |

The preceding panel has action codes typed in to view two transactions, delete one transaction, and update one transaction.

The fields on this panel provide the following information:

| | |
|---|---|
| **Corrid** | The correlation ID number of the transaction. |
| **Desc Flag** | The descriptor flag indicating the transaction's status. Valid descriptor flags are:<br>**R** Ready to be resent<br>**W** Waiting for response (sent but no response)<br>**F** Failed, no more retries will be attempted (but can be updated)<br>**D** Flagged for deletion |
| **Created Date** | The date the transaction was created. Dates are displayed in the format defined by the DATEFORMAT parameter in the initialization table. |
| **Created Time** | The time at which the transaction was created. |
| **Send Count** | The number of times the transaction has been sent to the dispatcher. |
| **Send Date** | The most recent date the transaction was sent to the dispatcher. Dates are displayed in the format defined by the DATEFORMAT parameter in the initialization table. |
| **Send Time** | The most recent time the transaction was sent to the dispatcher. |

| | |
|---|---|
| **OPID** | The ID of the operator station task (OST) that created the transaction. |
| **TRANSID** | The type of transaction that this is (IBCREATE, IBUPDATE, IBSEARCH). |
| **Database** | The label, as defined in the initialization table, of the database to which the transaction was sent. |
| **Bridge Dispatcher** | The name of the dispatcher that sent the transaction to the database server. |
| **Record Type** | The record type for the transaction. |

This is the panel that is displayed when you select a transaction for update. This panel is shown with values corresponding to the transaction selected for update in the panel on page 123.

```
EYLKUDUF            Update Checkpoint Record: CNM01@#X
    5697-SD9 (C) Copyright IBM Corp., 1981, 2001.
Edit control information, S=search/U=update options, parmvar names and
parmvar values.  Then press Enter.                               More:   +   >
 Dispatcher    Database      Record type    Transaction
 BRGDHIPR      INFONETW      PROBPP         IBCREATE

  SU  Alias,S-Word,VERIFIER,ASSOCDATA
 Opts Name+---10----+---20----+---30-- Data+---10----+---20----+---30----+---40
 S    NETWORK_NAME                     CNM01
      REPORTER_NAME                    AUTOBRIDGE
      DATE_OCCURRED                    02/28/1998
      TIME_OCCURRED                    15:50
 S    DEVICE_NAME                      L140A0F
  U   DESCRIPTION                      OPENED VIA NETVIEW BRIDGE BY SANDERS
      S0C09                            HARDWARE
  U   PROBLEM_STATUS                   OPEN
      CURRENT_PRIORITY                 04
      INITIAL_PRIORITY                 02
  U   DESCRIPTION_TEXT.1               THIS RECORD WAS CREATED VIA THE AUTOBR
  U   DESCRIPTION_TEXT.2               USING THE USER INPUT SCREEN SAMPLE

Command ==>
F1=Help F3=Exit F6=Roll F7=Backward F8=Forward F10=Left F11=Right F12=Cancel
```

This panel has four fields that display the control information for the specific transaction record displayed on the panel: Dispatcher, Database, Record type, and Transaction. You can change the values in these fields to display a different transaction record.

Below the control information, the panel displays a list of the fields specified in the *parmvar* variable of the API call that initiated this transaction. (The format of *parmvar* is described in “Handling Checkpoint Manager Transactions” on page 116.) You can modify any of the displayed values, and also add and delete fields from the list. The fields, their values, and any associated keyword operators are specified in three columns, as follows:

**SU Opts**
: This column indicates whether a SEARCH or UPDATE keyword is associated with this field. An S in this column indicates that the corresponding non-text alias or s-word is added to the searchlist for the transaction. A U in this column adds the alias name or s-word to the transaction’s updatelist. You cannot search or update ASSOCDATA and VERIFIER fields.

**Alias,S-Word,VERIFIER,ASSOCDATA**
: This column contains either an alias name such as REPORTER_NAME or DESCRIPTION_TEXT.1, an s-word such as S0C09 or S0E02.1 (not shown in the preceding example), the string ASSOCDATA, or the string VERIFIER. These items

are described in "ADD_DATA Function Syntax" on page 86, "VERIFIER Function Syntax" on page 87, and "ASSOCDATA Function Syntax" on page 87.

**Data** This column contains the value of the corresponding alias name, s-word, VERIFIER, or ASSOCDATA. This value can be up to 132 characters long. PF11 and PF10 enable you to scroll right and left, respectively, to view the entire 132-character field. Scrolling right and left shifts only the data portion of the row, leaving the first two columns in place.

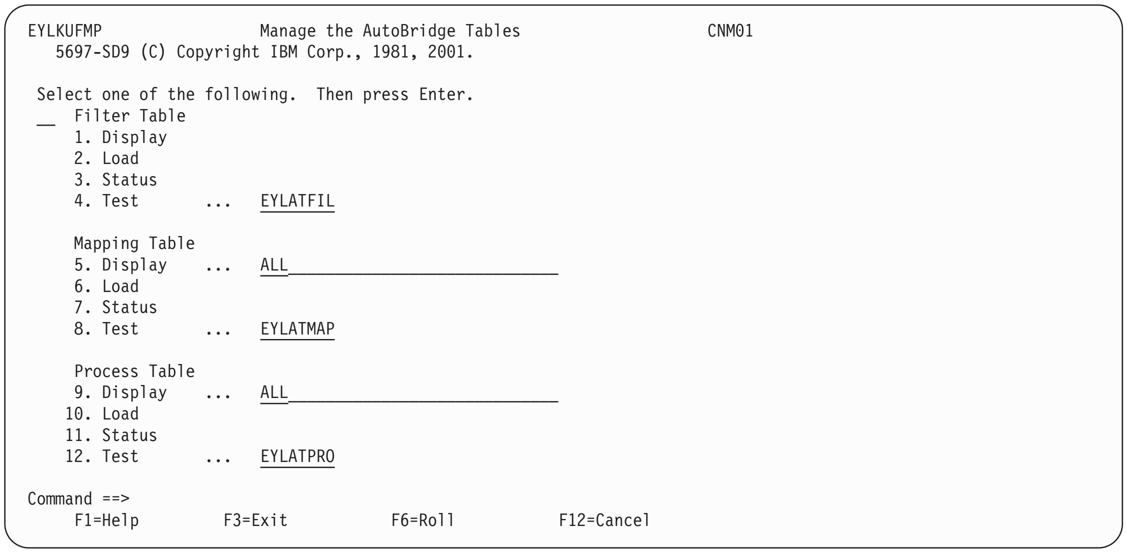
You can edit these values as follows:

- To replace an existing value in any of the three columns, type over it with a new value.
- To remove a field from the list, delete it using the Delete key or type over it with blanks.
- To add a new field to the list, type new data in the first available blank line.

When you have finished updating, deleting, or adding fields on the panel, press the Enter key to resubmit this transaction to the checkpoint manager. The transaction is then processed as though it were invoked via ABAPI or ABSUB.

## Managing the AutoBridge Tables

The ABTABLES command (EYLEUFMP), option 3 on the AutoBridge main menu, enables you to manage AutoBridge's process, mapping, and filter tables. You can browse a table, load it, or test its syntax. This is panel ABTABLES.



```
EYLKUFMP                 Manage the AutoBridge Tables                    CNM01
   5697-SD9 (C) Copyright IBM Corp., 1981, 2001.

 Select one of the following.  Then press Enter.
 __  Filter Table
     1. Display
     2. Load
     3. Status
     4. Test       ...   EYLATFIL

     Mapping Table
     5. Display    ...   ALL_____________________________
     6. Load
     7. Status
     8. Test       ...   EYLATMAP

     Process Table
     9. Display    ...   ALL_____________________________
    10. Load
    11. Status
    12. Test       ...   EYLATPRO

Command ==>
     F1=Help         F3=Exit          F6=Roll          F12=Cancel
```

You can display, load, test, or check the status of a table in one of the following ways:
- Type the number of the option and press Enter.
- Position the cursor next to the option and press Enter.

You can take the following actions:

**Display** Display the entire table or a specific segment you have selected.

**Load** Load the table you have selected into memory. All subsequent AutoBridge transactions will use the newly-loaded table.

| | |
|---|---|
| **Status** | Report whether the table is active, the time and date it was last loaded, and the user ID of the operator who loaded it. |
| **Test** | Test the syntax of the table. |

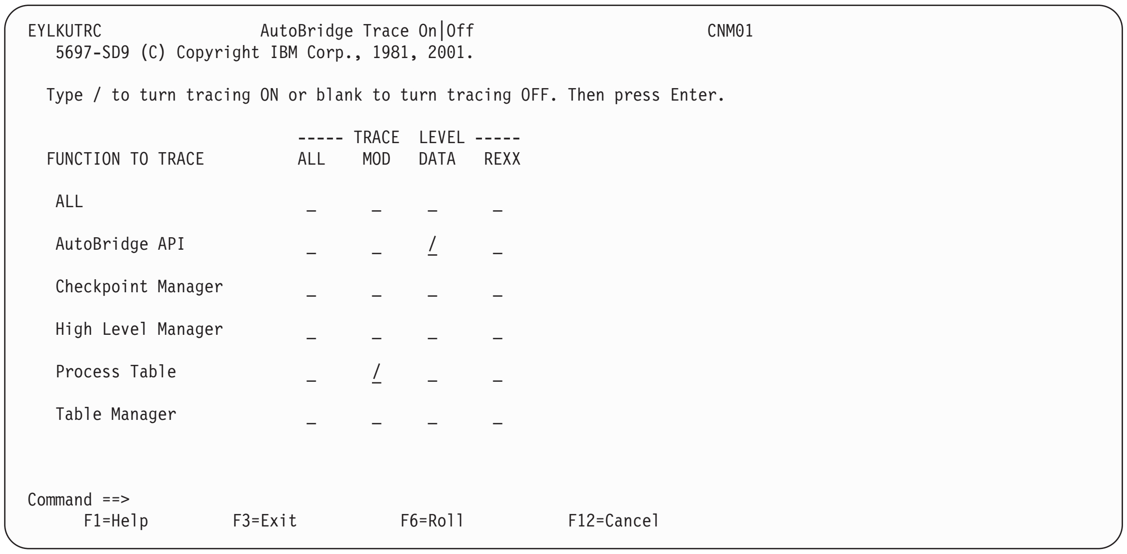
**Note:** The preceding panel displays the default names of the tables. You can overtype these default table names in order to Browse or Test another member in the DSIPARM data set. However, the Load and Status commands are valid only with one of the default table names:

| | |
|---|---|
| **Filter table** | EYLATFIL |
| **Mapping table** | EYLATMAP |
| **Process table** | EYLATPRO |

See "Coding NetView AutoBridge Tables" on page 81 for information on coding the process, mapping, and filter tables.

## Setting AutoBridge Tracing On or Off

The ABTRACE command (EYLEUTRC), option 4 on the AutoBridge main menu, enables you to turn AutoBridge tracing on or off. This is panel ABTRACE.



```
EYLKUTRC                AutoBridge Trace On|Off                        CNM01
   5697-SD9 (C) Copyright IBM Corp., 1981, 2001.

  Type / to turn tracing ON or blank to turn tracing OFF. Then press Enter.

                            ----- TRACE  LEVEL -----
  FUNCTION TO TRACE          ALL    MOD   DATA   REXX

   ALL                        _      _      _      _

   AutoBridge API             _      _      /      _

   Checkpoint Manager         _      _      _      _

   High Level Manager         _      _      _      _

   Process Table              _      /      _      _

   Table Manager              _      _      _      _


Command ==>
      F1=Help        F3=Exit          F6=Roll          F12=Cancel
```

This command enables, disables, and sets the parameters of AutoBridge's internal tracing function. AutoBridge allows multiple tracing levels for each of its component functions to aid in problem isolation.

To turn tracing on, choose a function to trace and enter a slash (/) below the appropriate trace level for that function. You can enable or disable tracing for all AutoBridge functions or select individual functions.

The available trace levels are:

**ALL** Enable all tracing options.
**MOD** Trace entry to and exit from the function.
**DATA** Trace transaction data through the function.
**REXX**
Issue the REXX language TRACE intermediate instruction. This option is valid for REXX program modules only.

The preceding example shows this panel with slashes that select both a Data level trace of the AutoBridge API and a Mod level trace of the process table.

To turn tracing off, delete the slash or type over it with a blank.

**Note:** You can also turn tracing on in the process table. However, tracing initiated by the process table is in effect only while the process table segment that invoked it is being accessed. All tracing initiated via this command (ABTRACE) is in effect until it is turned off via this command.

# 10

# NetView AutoBridge Implementation Scenarios

This chapter contains four scenarios to illustrate some of the ways that AutoBridge can function on your system. It includes scenarios that describe how to use AutoBridge to create records from the following types of data:

- BNJ146I data
- MSU data
- User-written application data
- Automated “unalert” notification data.

## BNJ146I Message Scenario

Suppose you want to open a record whenever an alert containing the text field of “APPLICATION ABEND” is received. You can extract the following data from the BNJ146I message:

NETWORK_NAME
PROBLEM_TYPE
DATE_OCCURRED
TIME_OCCURRED
ADDRESS_TEXT
DESCRIPTION_TEXT
RESOLUTION_TEXT

To allow you to track the origin and status of the problem, you might want to add the following constant data to the record:

PROGRAM_NAME
PROBLEM_STATUS
REPORTER_NAME
REPORTER_DEPT
REPORTER_PHONE
INITIAL_PRIORITY

You can add a NetView automation table entry that will invoke AutoBridge and specify a process table segment to be used. For example,

```
IF MSGID = 'BNJ146I' & TOKEN(4)= 'G' & TEXT= . 'APPLICATION ABEND' . THEN
 EXEC (CMD('ABAPI PROCESS_BNJ146G BRGDHIPR MSG')
```

This entry invokes AutoBridge and tells it to use the PROCESS_BNJ146G segment of the AutoBridge process table. It also specifies that the transaction is to be handled by the NetView Bridge dispatcher BRGDHIPR and that the input is a message.

The process table could be coded as:

```
BEGIN PROCESS_BNJ146G;
ADD_DATA 'INITIAL',PROBLEM_STATUS;
ADD_DATA 'NETVIEW',REPORTER_NAME,SEARCH;
ADD_DATA 'D15A',REPORTER_DEPT;
ADD_DATA '3',INITIAL_PRIORITY;
ADD_DATA '555-5430',REPORTER_PHONE;
ADD_DATA 'AUTOBRG',PROGRAM_NAME;
PARSE MAPPING=MAP_BNJ146G;
IBCREATE 'INFONETW','PROBLEM';
END PROCESS_BNJ146G;
```

Note that you could have coded the alias table with default values for REPORTER_NAME, and so on, rather than defining them in the process table.

To parse the data from the BNJ146I message, this segment calls the PARSE function specifying the MAP_BNJ146G segment in the AutoBridge mapping table:

```
PARSE MAPPING=MAP_BNJ146G;
```

The specified mapping table segment assigns data values from the input message keywords to specific alias names or s-words. It might look like the following example:

```
BEGIN MAP_BNJ146G;
MSGSTR(1,1) DATE_OCCURRED,EYLEXAYR(DATE_OCCURRED);
MSGSTR(2,1) TIME_OCCURRED;
TYPE(1,1) PROBLEM_TYPE,SEARCH;
DESC(1,1) DESCRIPTION_TEXT,TEXT,DECODE;
PC(1,1) DESCRIPTION_TEXT,TEXT,DECODE;
ACTS(1,1) RESOLUTION_TEXT,TEXT,DECODE;
USER(1,1) DESCRIPTION_TEXT,TEXT,DECODE;
FAIL(1,1) DESCRIPTION_TEXT,TEXT,DECODE;
INST(1,1) DESCRIPTION_TEXT,TEXT,DECODE;
TEXT(1) DESCRIPTION_TEXT,TEXT;
HIER(1,1) ADDRESS_TEXT,TEXT;
HIER(1,1) DEVICE_NAME,EYLEXHRL(N,L,DEVICE_NAME),SEARCH;
DOMID(1,1) NETWORK_NAME,SEARCH;
END MAP_BNJ146G;
```

Note that in this example, the REPORTER_NAME, PROBLEM_TYPE, and DEVICE_NAME are passed as SEARCH arguments.

This mapping table segment uses another keyword, DECODE, to specify that the input data is a code point and should be translated. Several fields within the BNJ146I message are code points and will be decoded so that their text can be added to the freeform text in the description section of the problem record.

This example also shows the use of the EYLEXAYR function, supplied with AutoBridge, which calculates the current year and adds it to the *MM/DD* date value taken from the message.

When the mapping segment finishes processing, it returns control to the process segment, where the IBCREATE statement that follows the PARSE statement is processed:

```
IBCREATE 'INFONETW','PROBLEM';
```

The IBCREATE function builds a create transaction using the field values just created by the process and mapping table segments. Because SEARCH data was specified, this transaction will be a conditional create, which means that if a duplicate record is found, a new record will not be created.

## MSU Scenario

Suppose you want to create a Tivoli Information Management for z/OS record based on an alert received from a local network. The alert comes to NetView in an NMVT, so you must first decide to select the NMVT for automation and processing by AutoBridge.

To select an MSU for automation, you can use the MSUSEG compare item. You can check for the existence of the major vector, subvectors, or subfields, or test the contents of a particular field. Refer to *NetView Customization: Writing Command Lists* for syntax.

For example, to test for the existence of an NMVT, you could place the following entry in the NetView automation table:

```
IF MSUSEG(0000) ¬= '' THEN
   EXEC (CMD('ABAPI PROCESS_GENALERT BRGDHIPR MSUSEG')
   ROUTE (ALL EYL));
```

If a generic alert major vector is detected, then the AutoBridge API is invoked with the following parameters:

- Process table segment = PROCESS_GENALERT
- Target dispatcher = BRGDHIPR
- Input type = MSUSEG

To check the contents of a particular field, you could code the following statement:

```
IF MSUSEG (0000.10(*)) = . 'IBM LAN NETWORK MANAGER' . THEN
   EXEC (CMD('ABAPI PROCESS_GENALERT BRGDHIPR MSUSEG')
   ROUTE (ALL EYL));
```

This entry invokes AutoBridge if any subvector X'10' (product set ID) contains the string “IBM LAN NETWORK MANAGER”. If you use an occurrence number instead of the asterisk (*), then only that subvector (X'10') could satisfy the comparison.

Now that you have modified the NetView automation table, the next step is to build the process table segment (PROCESS_GENALERT) that the API can use in processing the alert. As in the previous scenario, you can use the ADD_DATA function to add information not contained in the MSU so that you can track the Tivoli Information Management for z/OS record. This case adds data to these fields:

- PROBLEM_STATUS
- PROGRAM_NAME
- REPORTER_NAME
- REPORTER_DEPT
- REPORTER_PHONE
- INITIAL_PRIORITY

The process table segment for our generic alert may look like this:

```
***********************************************************************
*********     PROCESS TABLE SEGMENT FOR GENERIC ALERTS     ************
***********************************************************************
*
BEGIN PROCESS_GENALERT;
ADD_DATA 'INITIAL',PROBLEM_STATUS;
ADD_DATA 'NETVIEW',REPORTER_NAME,SEARCH;
ADD_DATA 'D15A',REPORTER_DEPT;
ADD_DATA '3',INITIAL_PRIORITY;
ADD_DATA '555-5430',REPORTER_PHONE;
ADD_DATA 'AUTOBRG',PROGRAM_NAME,SEARCH;
```

```
PARSE MAPPING=MAP_GENALERT,FILTER;
IBCREATE 'INFONETW','PROBLEM';
END PROCESS_GENALERT;
*
```

Notice that along with the ADD_DATA statements, there is a PARSE statement and an IBCREATE statement. The PARSE statement invokes the mapping table segment that contains statements for parsing the alert. Also note the FILTER keyword on the PARSE statement, indicating that AutoBridge will perform additional filtering as specified by the filter table.

The IBCREATE statement defines the name of the target database and the type of record being created.

To extract information from the NMVT, you need to code an appropriate segment in the mapping table. Mapping table segments for MSUs contain statements that define the location of discrete pieces of alert information, the alias name or s-word to which the information should be assigned, and other options described in “Coding NetView AutoBridge Tables” on page 81.

The mapping table segment for your generic alert might look like this:

```
*********************************************************************
******* THIS IS A MAPPING SEGMENT FOR A GENERIC ALERT ROUTED  *******
******* THROUGH THE NETVIEW AUTOMATION TABLE.                 *******
*********************************************************************
BEGIN MAP_GENALERT;
MSUSEG(0000.01.10,3,3) DATE_OCCURRED,EYLEXCDT('DATE_OCCURRED'X);
MSUSEG(0000.01.10,6,3) TIME_OCCURRED,EYLEXCTM('DATE_OCCURRED'X);
MSUSEG(0000.92,5,1) PROBLEM_TYPE,DECODE,SEARCH;
MSUSEG(0000.92,6,2) DESCRIPTION_TEXT,DECODE,UPDATE,TEXT;
MSUSEG(0000.93,3,2) DESCRIPTION_TEXT,DECODE,TEXT,UPDATE;
MSUSEG(0000.31.30,3) DESCRIPTION_TEXT,TEXT;
HIER(4) DEVICE_NAME,EYLEXNAM(DEVICE_NAME),SEARCH;
END MAP_GENALERT;
********************************************************************
```

The MSUSEG function statements in this segment, when parsed, do the following:

1. The first two statements retrieve the hexadecimal representations of the date and time from subvector X'01' (the date/time subvector). The EYLEXCDT function converts the hexadecimal representation of the date to text and assigns this value to the alias DATE_OCCURRED. The EYLEXCTM function does a similar conversion for the time and assigns that value to the alias TIME_OCCURRED. These functions are included in the AutoBridge samples library.
2. The third statement extracts the alert type code point, decodes it, and assigns the text to the alias name PROBLEM_TYPE. The SEARCH option specified on this statement specifies that the PROBLEM_TYPE value is used in duplicate records searches.
3. The fourth statement extracts the alert description code point, decodes it, and assigns the text to the DESCRIPTION_TEXT alias name. The UPDATE keyword on the statement indicates that the alias DESCRIPTION_TEXT is used to update the duplicate record, if one is found.
4. The fifth statement extracts the probable cause code point, decodes it, and appends the text to DESCRIPTION_TEXT. In both statements 4 and 5, the TEXT keyword specifies that DESCRIPTION_TEXT is a freeform text field and should be appended to the variable TEXTLIST in the *parmvar* list.

5. The sixth statement extracts the self-defining text from subvector X'30' and appends it to DESCRIPTION_TEXT. If subvector X'30' was not in the NMVT, AutoBridge generates message EYL159W.
6. The final statement extracts the fourth name/type pair from the hierarchy list and invokes the function EYLEXNAM, which strips the type data. It assigns this name value to the alias DEVICE_NAME.

Since our process table segment for this alert specified AutoBridge filtering, you can code a filter table to look like this:

```
*********************************************************************
*     AUTOBRIDGE FILTER TABLE SAMPLE
*********************************************************************
DEFAULT BLOCK;
PROBLEM_TYPE=PERM & DEVICE_NAME=LAN*;
*
```

In this table, the first statement defines a default filtering value of BLOCK. Because of this, all inputs will be blocked from AutoBridge processing unless the second statement is true. In this example, the NMVT being processed will result in a Tivoli Information Management for z/OS record being created or updated only if PROBLEM_TYPE=PERM and DEVICE_NAME=“any value starting with ‘LAN’”.

Once the AutoBridge tables are in place, you can activate them by either recycling the AutoBridge application or by manually reloading those tables that were changed. See “Managing the AutoBridge Tables” on page 125. You must also reload the NetView automation table. You can use the AUTOTBL command to do this. Its syntax is AUTOTBL MEMBER=*membername*.

You should also make sure that the NPDA filters are set to pass alerts to the NetView automation table. Refer to *NetView Operation* for information on doing this.

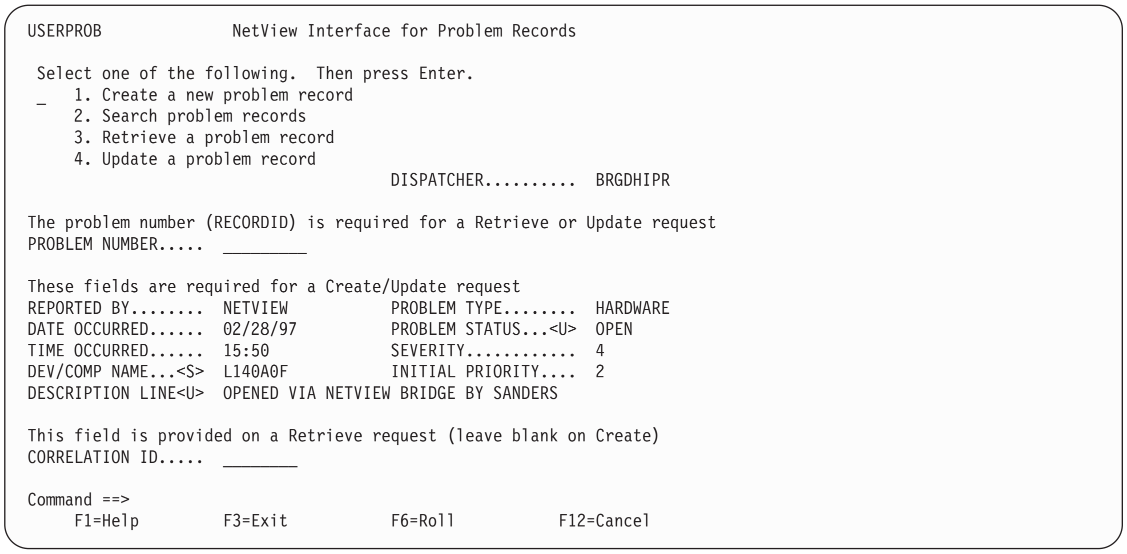
## User-Written Application Data Scenario

This scenario demonstrates how you can use AutoBridge to create Tivoli Information Management for z/OS records with data passed from a user-written application. For this example, suppose there’s an automation exec on your system that is attempting to vary a device online but is unsuccessful. This exec provides the option of creating a problem record. To do so, you can put the name of the device and any other pertinent information in task global variables and call your user-written application to invoke the NetView Bridge.

The originating exec could include something like this:

```
If operator_option = 'info_record' then Do
  DEVICE_NAME = vary_net
  'GLOBALV PUTT DEVICE_NAME'
  'USERPROB'
End
```

This code fragment invokes an exec named USERPROB. You can customize the sample exec EYLEXUSR, supplied in the AutoBridge sample library, to create such an exec. USERPROB displays a panel that allows you to create, search, retrieve, and update problem records. This panel is pre-filled with default and task global values that you can use, add to, modify, or delete, as shown in the following example:



```
USERPROB             NetView Interface for Problem Records

 Select one of the following.  Then press Enter.
 _   1. Create a new problem record
     2. Search problem records
     3. Retrieve a problem record
     4. Update a problem record
                                       DISPATCHER.........  BRGDHIPR

The problem number (RECORDID) is required for a Retrieve or Update request
PROBLEM NUMBER.....  _________

These fields are required for a Create/Update request
REPORTED BY........  NETVIEW           PROBLEM TYPE........  HARDWARE
DATE OCCURRED......  02/28/97          PROBLEM STATUS...<U>  OPEN
TIME OCCURRED......  15:50             SEVERITY............  4
DEV/COMP NAME...<S>  L140A0F           INITIAL PRIORITY....  2
DESCRIPTION LINE<U>  OPENED VIA NETVIEW BRIDGE BY SANDERS

This field is provided on a Retrieve request (leave blank on Create)
CORRELATION ID.....  ________

Command ==>
     F1=Help         F3=Exit           F6=Roll           F12=Cancel
```

Pressing the Enter key on this panel performs the specified action using the displayed values.

USERPROB bypasses the AutoBridge API (ABAPI) by formatting the NetView Bridge record and invoking the AutoBridge checkpoint manager (ABSUB) directly. No process table, mapping table, or filter table data is referenced in this transaction.

In this example, the data is passed as follows:

```
'ABSUB BRGDHIPR INFONETW PROBLEM IBCREATE PARMVAR'
```

where:

```
PARMVAR =  NETWORK_NAME REPORTER_NAME DATE_OCCURRED TIME_OCCURRED
DEVICE_NAME DESCRIPTION PROBLEM_TYPE PROBLEM_STATUS CURRENT_PRIORITY
INITIAL_PRIORITY SEARCHLIST UPDATELIST
```

and where:

```
NETWORK_NAME = CNM01
REPORTER_NAME = NETVIEW
DATE_OCCURRED = 02/28/1997
TIME_OCCURRED = 15:50
DEVICE_NAME = L140A0F
DESCRIPTION = OPENED VIA NETVIEW BRIDGE BY SANDERS
PROBLEM_TYPE = HARDWARE
PROBLEM_STATUS = OPEN
CURRENT_PRIORITY = 4
INITIAL_PRIORITY = 2
SEARCHLIST = EYLLIST NETWORK_NAME DEVICE_NAME
UPDATELIST = EYLLIST DESCRIPTION PROBLEM_STATUS
```

A search is performed on the NETWORK_NAME and DEVICE_NAME. If no match is found, a new record is created. If a match is found, the DESCRIPTION line and PROBLEM_STATUS fields are updated. The panel displays the resulting correlation ID and problem number for reference.

## Automated “Unalert” Notification Scenario

An alert major vector might flow into NetView with a description code point (subvector 92) of X'A000' (PROBLEM RESOLVED) or X'A001' (IMPENDING COOLING PROBLEM RESOLVED). You might want to conditionally update any open problem records to let the technician assigned to work on the problem know that this “unalert” has been received.

You can accomplish this by including an entry in the NetView automation table to trap either the MSU or BNJ146I message. The MSU entry would look like this:

```
IF MSUSEG(0000.92 6 ) = HEX('A0') .
   & HIER = DEVICE THEN
   EXEC (CMD('AUNALERT 'DEVICE) ROUTE (ONE AUTOBR1 *));
```

The BNJ146I entry would look like this:

```
IF MSGID='BNJ146I' & TEXT = 'DESC=A' .
   & TEXT = . 'HIER=' HIER . THEN
   EXEC (CMD('AUNALERT 'HIER) ROUTE (ONE AUTOBR1 *));
```

These table entries invoke an exec called AUNALERT, which is based on the sample exec EYLEXA00 provided with AutoBridge. AUNALERT calls the AutoBridge API with a search request for OPEN problem records in this domain for the device name parsed from the NetView automation table. The AutoBridge API is called as follows:

```
'ABAPI SEARCH_FOR BRGDHIPR none'
```

The process table segment that performs this request would be:

```
BEGIN SEARCH_FOR;
GLOBALV GETT DEVICE;
ADD_DATA DEVICE,DEVICE_NAME,SEARCH;
ADD_DATA NVID(),NETWORK_NAME,SEARCH;
ADD_DATA 'OPEN',PROBLEM_STATUS,SEARCH;
ASSOCDATA DESCRIPTION;
IBSEARCH 'INFONETW','PROBLEM';
END SEARCH_FOR;
```

The AUNALERT exec traps the response messages from the NetView Bridge search request. If one or more record IDs are returned, the highest problem record ID with a record processing code of ‘00’ (no error detected) is updated with an informative DESCRIPTION line such as the following:

```
new_desc = 'THIS ALERT HAS BEEN RESOLVED -- SEE NETVIEW'
'GLOBALV PUTT new_desc'
```

The AutoBridge API is called as follows:

```
'ABAPI UPDATE_IT BRGDHIPR none'
```

The process table segment UPDATE_IT would look like this:

```
BEGIN UPDATE_IT;
GLOBALV GETT HIGH_RECID;
GLOBALV GETT NEW_DESC;
ADD_DATA HIGH_RECID,RECORDID;
ADD_DATA NEW_DESC,DESCRIPTION,UPDATE;
VERIFIER REPORTER_NAME=NETVIEW;
IBUPDATE 'INFONETW','PROBLEM';
END UPDATE_IT;
```

When the REPORTER_NAME value is NETVIEW, then this segment updates the Tivoli Information Management for z/OS OPEN problem record for this device, in this domain, with the new DESCRIPTION text.

11

# NetView AutoBridge Planning

This chapter describes all the planning steps necessary to implement NetView AutoBridge. Complete and verify the steps listed here before the start of implementation.

Use this table to keep track of your completed work:

*Table 24. AutoBridge planning checklist*

| ✓ | **Step** |
|---|---|
| | Step 1. Verify Installation of Required Hardware |
| | Step 2. Verify Installation of Required Software |
| | Step 3. Verify Required Skills and Documentation Present |
| | Step 4. Choose an Application ID and Receive a Queue Name |
| | Step 5. Plan the Initialization Table |
| | Step 6. Plan the NetView Automation Table Customization |
| | Step 7. Plan the PIDT, PIPT, and Alias Table Modifications |
| | Step 8. Plan the Process Table |
| | Step 9. Plan the Mapping Table |
| | Step 10. Plan the Filter Table |

## Step 1. Verify Installation of Required Hardware

Make sure that the following hardware is present on your host system:

- One 3480 or 3490 tape drive for the installation of the cartridge (6250)
- Sufficient DASD storage as specified in the Tivoli Information Management for z/OS NetView AutoBridge section of the Tivoli Information Management for z/OS Program Directory

## Step 2. Verify Installation of Required Software

Make sure that the following software is present on your host system:

- NetView Version 3 for MVS/ESA (5655–007) or a subsequent release

## Step 3. Verify Required Skills and Documentation Present

You should have the following skills, knowledge, and documentation present before installing AutoBridge.

- Skill in using JCL to implement programs
- Skill in creating high-level qualifiers
- Skill in creating VSAM data sets
- Skill in using the Tivoli Information Management for z/OS Panel Modification Facility (PMF)
- Skill in assigning NetView messages to operator IDs
- Skill in coding NetView automation table entries
- Knowledge of NetView alert and message formats
- Knowledge of the data sets in which your Tivoli Information Management for z/OS and NetView programs reside
- Access to the following books:
  - *NetView Installation and Administration Guide--(SC31-6051)*
  - *NetView Automation Implementation--(LY43-0008)*
  - *Tivoli Information Management for z/OS User's Guide*
  - *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*
  - *Tivoli Information Management for z/OS Program Administration Guide and Reference*
  - *Tivoli Information Management for z/OS Panel Modification Facility Guide*

## Step 4. Choose an Application ID and Receive a Queue Name

The NetView Bridge adapters require an application identifier (APPLID) specified in the startup JCL, as well as a unique receive queue identifier to be used by the PPI. The example on page 8 shows a sample of the JCL and an explanation of these fields is in "The EXEC Statement" on page 9.

**Note:** ALL STEPLIB data sets must be APF-authorized.

## Step 5. Plan the Initialization Table

Use the information in "Coding the Initialization Table" on page 101 to complete a copy of the initialization table worksheet on page 231 for each NetView on your system that will use AutoBridge. Use the initialization table worksheets and the sample member EYLATINT to code an initialization table for each NetView.

If your system handles a high volume of transactions, be careful in the assignment of the NetView Bridge dispatchers and adapters. The configuration of these components is critical to the level of performance achieved on your system.

## Step 6. Plan the NetView Automation Table Customization

You may need to customize your NetView automation table for use with AutoBridge. Plan entries for the following:

- An entry to automatically start AutoBridge during NetView startup
- An entry to trap all AutoBridge messages and route them to all operators in the AutoBridge group
- Entries for all MSUs or messages that will drive the AutoBridge API.

The NetView automation table intercepts system messages and all alerts and MSUs that are not filtered by NetView’s hardware monitor (formerly called NPDA). You can use the NetView automation table to parse and filter input, and then pass it to AutoBridge via a call to the AutoBridge API. You can also use the NetView automation table to pass input to user-written routines or command processors that, in turn, invoke AutoBridge.

The following example shows a NetView automation table entry that invokes AutoBridge:

```
IF MSUSEG(0000.92 5 1) = '01' & MSUSEG('0000.10.(2)') = .'IBM LAN MANAGER'.
  EXEC (CMD('ABAPI LAN_MSU BRGDHIPR MSUSEG'));
```

This entry produces the following results:

```
If a generic alert specifies a permanent error (MSUSEG(0000.92 5 1) = ’01 ’)
and
if the alert is from IBM LAN Manager (as specified in the second instance of
                subvector 10 in the NMVT)
then
  execute the AutoBridge API along with a call to the LAN_MSU process
                table segment
  and process the transaction via the dispatcher task BRGAHIPR
  and use the MSUSEG as input to the AutoBridge API.
```

For more information on invoking AutoBridge, see “Invoking AutoBridge” on page 114.

For an overview of how to define automated message statements, refer to *NetView Installation and Administration Guide* For the syntax of NetView automation table statements, refer to *NetView Administration Reference*.

## Step 7. Plan the PIDT, PIPT, and Alias Table Modifications

Use the information in the *Tivoli Information Management for z/OS Application Program Interface Guide* to determine which of these tables must be customized. If you have customized Tivoli Information Management for z/OS, you may need to add or modify those fields that will be passed to the Tivoli Information Management for z/OS High Level API (HLAPI) via AutoBridge.

The PIDT defines the data that passes between AutoBridge and the Tivoli Information Management for z/OS High Level API. The Program Interface Pattern Table (PIPT) enables applications to test record or search argument data against validation patterns. The alias table (PALT) lets you use external symbolic names (alias names) for data fields, define default values, and specify which fields are required.

Both the AutoBridge initialization table and the user-supplied input data set INFOBRDS contain entries that affect the use of the PIDT, PIPT, and alias tables. At this time, you may want to consider the values you will assign to these entries. Table 25 summarizes each of these entries.

Data view records can be used instead of PIDTs. To use data view records, specify BYPASS_PANEL_PROCESSING=YES in the INFOBRDS data set. If BYPASS_PANEL_PROCESSING=NO is specified, data view records can still be used if the INFOBRDS keyword parameter USE_DATA_VIEW is specified as YES.

"Coding the Initialization Table" on page 101 describes the contents of the AutoBridge initialization table. "The User-Supplied Input Data Set" on page 10 discusses creating the INFOBRDS data set.

*Table 25. Environment settings relative to PIDT, PIPT, and alias tables*

| Table | Entry | Purpose |
|---|---|---|
| EYLATINT (initialization table) | VOCAB | Defines the alias table to be used. An alias table created by BLGUT8 is supplied in member EYLALIA of SBLMFMT. The source is member EYLALIAS, supplied in SBLMSAMP.<br><br>The VOCAB entry in the initialization table specifies the name of the PALT containing alias names, defaults, and other entries. |
| | IBCREATE, IBUPDATE, IBSEARCH, INQVIEW | Specifies the name of the PIDT to be used for each specific transaction. Values for these PIDTs are supplied for the Tivoli default record types. If BYPASS_PANEL_PROCESSING=YES is specified in the INFOBRDS data set, then the values specified for IBCREATE, IBUPDATE, IBSEARCH, and INQVIEW will be used as data view names. Even if BYPASS_PANEL_PROCESSING=NO is specified in the INFOBRDS data set, if USE_DATA_VIEW=YES is specified, these values will be used as data view names. |
| INFOBRDS (input data set) | DEFAULT_OPTION | Specifies how default processing of alias entries is handled.<br><br>You may define default values and specify whether a field is required for each entry in an alias table. If this option is set to ALL, the defaults you defined are used if no other value is specified. If set to REQUIRED, only required fields assume their default value if none is specified. If set to NONE, no default values are used. |
| | VALIDATE | Determines if the HLAPI performs data response validation when processing input.<br><br>If VALIDATE is set to YES, the HLAPI will access the PIPT associated with the specified PIDT. If set to NO, no validation is performed.<br>**Note:** A VALIDATE setting of NO may cause unexpected records to appear in the database. |

## Step 8. Plan the Process Table

Use the information in “Coding the Process Table” on page 83 to complete a copy of the process table worksheet on page 233. Use the sample member EYLATPRO to aid in completing the process table worksheet.

## Step 9. Plan the Mapping Table

Use the information in “Coding the Mapping Table” on page 92 to complete a copy of the mapping table worksheet on page 234. Use the sample member EYLATMAP to aid in completing the mapping table worksheet.

## Step 10. Plan the Filter Table

Use the information in “Coding the Filter Table” on page 97 to complete a copy of the filter table worksheet on page 235. Use the sample member EYLATFIL to aid in completing the filter table worksheet.

# NetView AutoBridge Software Setup and Administration

This chapter describes the activities you must perform for AutoBridge to function on your system. It is divided into two sections:

- "Setting Up the Resident NetView" on page 144 describes the steps for setting up the NetView Bridge and AutoBridge components on the resident NetView as required by AutoBridge.
- "Setting Up Remote NetViews" on page 153 describes the steps for setting up the remote NetView Bridge and the remote installation of AutoBridge.

For directions on installing the AutoBridge program, refer to the Tivoli Information Management for z/OS NetView AutoBridge section of the Tivoli Information Management for z/OS Program Directory.

To give you a better idea of the NetView Bridge and AutoBridge components that you must define in these setup procedures, Figure 10 on page 144 shows a diagram of the relationship between NetView Bridge adapters, NetView Bridge dispatchers, checkpoint manager tasks, and remote dispatchers.

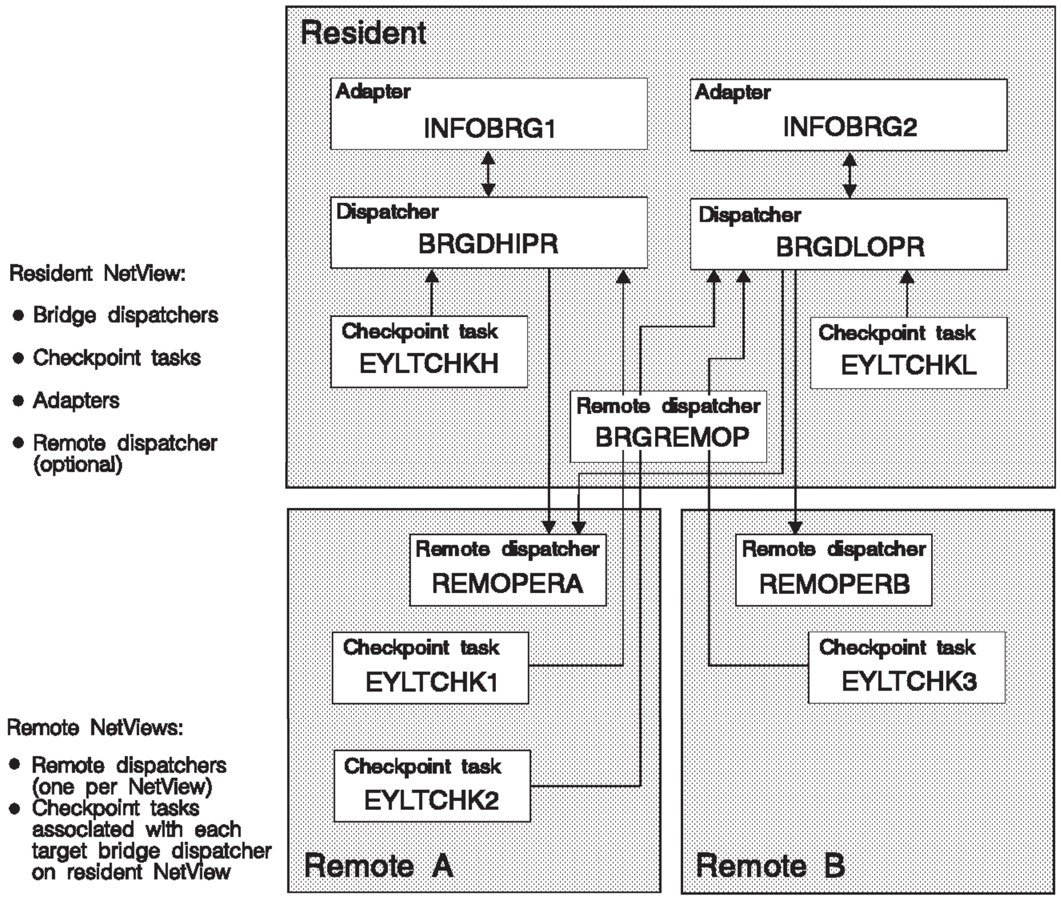


*Figure 10. Relationship of NetView autotasks in AutoBridge process flow*

## Setting Up the Resident NetView

Follow these steps to set up NetView Bridge and AutoBridge components on the resident NetView:

*Table 26. Resident NetView setup checklist*

| ✓ | Step |
|---|---|
| | Add operator IDs to NetView for the bridge dispatcher, checkpoint manager, and remote bridge dispatcher (optional) tasks as described in “Adding Operator IDs for NetView Autotasks” on page 145. |
| | Create autotask profiles for each new operator ID as described in “Creating Profiles for NetView Autotasks” on page 146. |
| | Add NetView Bridge and AutoBridge command model statements to NetView as described in “Adding Command Model Statements to NetView” on page 148. |
| | Modify the NetView procedure JCL to include command list, panel, message, and STEPLIB data sets as described in “Modifying the NetView Procedure JCL” on page 149. |
| | Modify the JCL supplied with the Tivoli Information Management for z/OS NetView Bridge Adapter as described in “Modifying the JCL Supplied with the Tivoli Information Management for z/OS NetView Bridge Adapter” on page 150. |

*Table 26. Resident NetView setup checklist (continued)*

| ✓ | Step |
|---|---|
| | Customize the DSIDMN member of the concatenated DSIPARM data set as described in "Customizing the DSIPARM DSIDMN Member" on page 151. |
| | Allocate a VSAM data set for the AutoBridge checkpoint files as described in "Allocating the Checkpoint File VSAM Data Set" on page 150. |
| | Add the application identifier to the proper class definition as described in "Modifying the JCL Supplied with the Tivoli Information Management for z/OS NetView Bridge Adapter" on page 150. |
| | Create any additional copies of the NetView Bridge adapter server you need to improve performance of the bridge as described in "Creating Additional Copies of a Server" on page 151. |
| | Customize the Tivoli Information Management for z/OS PIDT, PIPT, and alias tables as described in "Customizing the PIDT, PIPT, and Alias Tables" on page 152. |
| | Customize the NetView automation table as described in "Customizing the NetView Automation Table" on page 152. |
| | Customize the initialization table as described in "Creating the Initialization Table" on page 153. |
| | Create the process table as described in "Creating the Process Table for Resident NetView" on page 153. |
| | Create the mapping table as described in "Creating the Mapping Table for Resident NetView" on page 153. |
| | Create the filter table as described in "Creating the Filter Table for Resident NetView" on page 153. |

## Adding Operator IDs for NetView Autotasks

You must define at least one operator ID for each of the following autotasks:
- NetView Bridge dispatchers
- Checkpoint managers (one for each NetView Bridge dispatcher)
- Remote dispatcher (optional)

The function of these autotasks can be summarized as follows:

| | |
|---|---|
| **NetView Bridge dispatcher** | Establishes the NetView Bridge environment |
| **Checkpoint manager** | Controls transaction flow through its associated dispatcher |
| **Remote dispatcher** | Provides remote access to the bridge |

When defining these autotasks, be aware of the following conventions:

- The resident NetView can have only one remote dispatcher.
- For performance reasons, you may define multiple bridge dispatchers, each handling different types of traffic.
- Each bridge dispatcher may communicate with one or more copies of a database server.
- Your autotasks should be exclusive to the NetView Bridge.

To define the IDs, you must modify the DSIOPF member of DSIPARM. The member EYLRES of the sample data set SEYLSPL contains the following operator ID definitions that you can copy into DSIOPF and modify as needed:

```
BRGREMOP     OPERATOR    PASSWORD=BRGREMOP
             PROFILEN    EYLPRFRD
BRGDHIPR     OPERATOR    PASSWORD=BRGDHIPR
             PROFILEN    EYLPRFDH
EYLTCHKH     OPERATOR    PASSWORD=EYLTCHKH
             PROFILEN    EYLPRFAO
BRGDLOPR     OPERATOR    PASSWORD=BRGDLOPR
             PROFILEN    EYLPRFDL
EYLTCHKL     OPERATOR    PASSWORD=EYLTCHKL
             PROFILEN    EYLPRFAO
```

This sample includes operator ID definitions for high and low priority NetView Bridge dispatchers (BRGDHIPR and BRGDLOPR), their associated checkpoint managers (EYLTCHKH and EYLTCHKL), and a remote dispatcher (BRGREMOP). Note that each definition contains a pointer to an associated profile. The following section describes how to create these profiles.

Your ID definitions take effect the next time NetView is recycled.

## Creating Profiles for NetView Autotasks

You must create new members in a data set of NetView DSIPRF concatenation having the profile names referenced in the operator ID definitions you created in the previous section.

These members authorize the operator ID to act as a NetView Bridge autotask. The profile for the NetView Bridge dispatcher specifies the NetView Bridge command RTRINIT as the initial command that runs at task initialization time to start the NetView Bridge. The profile for the remote dispatcher specifies the NetView Bridge command REMOTEBR to initialize remote access to the bridge.

By customizing the sample profiles provided with AutoBridge (EYLPRFAO, EYLPRFDH, EYLPRFDL, and EYLPRFRD), you can create profiles for the NetView Bridge dispatchers, their associated checkpoint manager tasks, and, optionally, a remote dispatcher.

The following are examples of PROFILE statements:

```
EYLPRFDH   PROFILE   IC=RTRINIT,HOLDQ01,READYQ01,OUTPTQ01,2000
           AUTH      MSGRECVR=NO,CTL=GLOBAL
           OPCLASS   1,2,6
           END

EYLPRFDL   PROFILE   IC=RTRINIT,HOLDQ02,READYQ02,OUTPTQ02,200
           AUTH      MSGRECVR=NO,CTL=GLOBAL
           OPCLASS   1,2
           END

EYLPRFRD   PROFILE   IC=REMOTEBR
           AUTH   MSGRECVR=NO,CTL=GLOBAL
           OPCLASS 1,2,6
           END

EYLPRFAO   PROFILE
           AUTH   CTL=GLOBAL
           OPCLASS 1,2
           END
```

The OUTPTQ*nn* value must match the SEND_QUEUE='OUTPTQ*nn*' setting defined in the Tivoli Information Management for z/OS NetView Bridge Adapter INFOBRDS input data set as shown in the example on page 11.

For information on the RTRINIT and REMOTEBR commands, see “RTRINIT Command” and “REMOTEBR Command”.

## RTRINIT Command

NetView Bridge uses the RTRINIT command to activate the interface between a NetView autotask and a specific set of database servers. This command must be executed under an autotask. See “Adding Operator IDs for NetView Autotasks” on page 145 for instructions on creating this autotask.

The RTRINIT command is driven by the profile of a NetView autotask. This autotask serves as the interface to a specific set of database servers and connects three user-provided queue names to the NetView program-to-program interface (PPI). A bridge dispatcher executes the NetView Bridge command RTRINIT to establish the NetView Bridge environment. For more information on PPI queues, refer to the *NetView Application Programming Guide*.

**Note:** The parameters described in the following syntax are positional and must be placed in the order shown.

The syntax for the RTRINIT command is:

```
►►──RTRINIT──hqueue,──rqueue,──oqueue,──hqueuel──────────────────────►◄
```

where:

*hqueue*
: Is the name of the hold queue. A PPI queue is created to save transactions that have not been dispatched to a database server. This 1- to 8-character field is required and can contain uppercase alphabetic characters, numeric characters, or $, %, &, @, and #.

*rqueue*
: Is the name of the ready queue. A PPI queue is created to save the READY tokens generated by the database servers. This 1- to 8-character field is required and can contain uppercase alphabetic characters, numeric characters, or $, %, &, @, and #.

*oqueue*
: Is the name of the output queue. A PPI queue is created to receive transaction replies and control messages from the database servers. This queue has the same name as the send queue name defined in INFOBRDS as shown in the example on page 11.

  This 1- to 8-character field is required and can contain uppercase alphabetic characters, numeric characters, or $, %, &, @, and #.

*hqueuel*
: Defines the limit of the HOLD queue to the PPI. This parameter is required and must be an integer. Its value must be between 50 and 2000 inclusive.

## REMOTEBR Command

The REMOTEBR command enables the remote dispatcher to register as a management services application on the high-performance transport API. This command is driven by the profile of the remote dispatcher autotask. The autotask must be running on both the remote NetView that is sending transactions and receiving replies and on the resident NetView in which the database server resides. Only one autotask in each NetView can issue this command at a time. Consecutive invocations of this command by other tasks results in message DWO534I being issued.

**Note:** Although you can execute REMOTEBR from the same autotask NetView Bridge is using to communicate with a resident database server, this setup is not recommended. Issue the REMOTEBR command from a new autotask. See “Adding Operator IDs for NetView Autotasks” on page 145 for instructions on creating this autotask.

The syntax for the REMOTEBR command is:

```
►►──REMOTEBR──────────────────────────────────────────────────────────►◄
```

## Adding Command Model Statements to NetView

You must define the NetView Bridge and AutoBridge commands to NetView by using command model (CMDMDL) statements. These statements reside in the DSICMD member of the DSIPARM data set.

To define these statements for the NetView Bridge, remove the asterisks that comment out the following command model statements in the DSICMD member:

```
RTRINIT   CMDMDL MOD=DSINBINT,TYPE=R,RES=N
RTRQUEUE  CMDMDL MOD=DSINBQUE,TYPE=R,RES=Y
TRANRCV   CMDMDL MOD=DSINBRCV,TYPE=R,RES=Y
TRANSND   CMDMDL MOD=DSINBSND,TYPE=R,RES=Y
DSINBRSM  CMDMDL MOD=DSINBRSM,TYPE=R,RES=Y
DSINBTRM  CMDMDL MOD=DSINBTRM,TYPE=R,RES=Y
```

For remote bridge operation, the following command model statements are also required:

```
DSINBR62  CMDMDL MOD=DSINBR62,TYPE=R,PARSE=N,RES=Y
REMOTEBR  CMDMDL MOD=DSINBREM,TYPE=R,PARSE=Y,RES=Y
DSINBRLG  CMDMDL MOD=DSINBRLG,TYPE=R,PARSE=Y,RES=Y
```

To define command model statements for AutoBridge, copy the member EYLCMD from the AutoBridge-supplied SEYLSPL data set into the DSICMD member. Figure 11 on page 149 shows the contents of EYLCMD.

```
*-----------------------------------------------------------------*
*  AUTOBRIDGE                                                     *
*  DESCRIPTION: COMMAND MODEL ENTRIES FOR THE NV AUTOBRIDGE       *
*-----------------------------------------------------------------*
EYLEAPI    CMDMDL   MOD=DSICCP,ECHO=N
           CMDSYN   ABAPI
EYLEHBRG   CMDMDL   MOD=DSICCP,ECHO=N
           CMDSYN   ABRIDGE
EYLEUCKP   CMDMDL   MOD=DSICCP,ECHO=N
           CMDSYN   ABCHECKP
EYLEUMEN   CMDMDL   MOD=DSICCP,ECHO=N
           CMDSYN   ABMENU
EYLEUFMP   CMDMDL   MOD=DSICCP,ECHO=N
           CMDSYN   ABTABLES
EYLEUSRS   CMDMDL   MOD=DSICCP,ECHO=N
           CMDSYN   ABSRS
EYLEUTRC   CMDMDL   MOD=DSICCP,ECHO=N
           CMDSYN   ABTRACE
EYLSTMEM   CMDMDL   MOD=EYLSTMEM,TYPE=R,RES=N,ECHO=Y
           CMDSYN EYLSTMEM
           CMDSYN EYLMEM
EYLSSVAR   CMDMDL   MOD=EYLSSVAR,TYPE=R,RES=Y
EYLSMSG    CMDMDL   MOD=EYLSMSG,ECHO=N,TYPE=R,RES=Y
EYLSCSUB   CMDMDL   MOD=EYLSCSUB,TYPE=R,RES=N,PARSE=Y
           CMDSYN ABSUB
EYLSCGET   CMDMDL   MOD=EYLSCGET,TYPE=R,RES=N,PARSE=Y
EYLSCDEL   CMDMDL   MOD=EYLSCDEL,TYPE=R,RES=N,PARSE=Y
EYLSCLST   CMDMDL   MOD=EYLSCLST,TYPE=R,RES=N,PARSE=Y
EYLSCVSM   CMDMDL   MOD=EYLSCVSM,TYPE=R,RES=N,PARSE=Y
EYLSCIMT   CMDMDL   MOD=EYLSCIMT,TYPE=R,RES=N,PARSE=Y
EYLSCIRC   CMDMDL   MOD=EYLSCIRC,TYPE=R,RES=N,PARSE=Y
EYLSCPCF   CMDMDL   MOD=EYLSCPCF,TYPE=R,RES=N,PARSE=Y
EYLSMVSM   CMDMDL   MOD=EYLSMVSM,TYPE=D,RES=N,PARSE=N
EYLSCLPR   CMDMDL   MOD=EYLSCLPR,TYPE=D,RES=N,PARSE=N
EYLSTMGR   CMDMDL   MOD=EYLSTMGR,TYPE=R,RES=N,ECHO=Y
EYLTMGR    CMDMDL   MOD=EYLSTSUB,TYPE=D,RES=N,PARSE=N
EYLSTNCP   CMDMDL   MOD=EYLSTNCP,TYPE=R,RES=N,PARSE=N
****** END AUTOBRIDGE COMMANDS *****
```

*Figure 11. AutoBridge sample command model statement definitions*

## Modifying the NetView Procedure JCL

Modify the NetView startup procedure JCL to include the AutoBridge data sets. You must add these statements to the command list, panel, message, and STEPLIB data set concatenation lists. If you have changed the data set allocation statements or renamed the data sets after they were unloaded, use the new names that you have chosen.

**Command lists**

Your command list data set concatenation list should contain the following lines:

```
//DSICLD DD DSN=current clists
//       DD DSN=EYL.SEYLCL1
//       DD DSN=EYL.SEYLSPL
```

**Panels**

Your panel data set concatenation list should contain the following lines:

```
//CNMPNL1 DD DSN=current panels
//        DD DSN=EYL.SEYLPN1,DISP=SHR
//        DD DSN=EYL.SEYLPN2,DISP=SHR
```

**Messages**

Your message data set concatenation list should contain the following lines:

```
//DSIMSG  DD DSN=current messages
//        DD DSN=EYL.SEYLMSG,DISP=SHR
```

**STEPLIB**

Your STEPLIB data set concatenation list should contain the following lines:

```
//STEPLIB DD DSN=current steplibs
//        DD DSN=EYL.SEYLMOD
```

**Note:** All libraries in this STEPLIB must be authorized.

## Modifying the JCL Supplied with the Tivoli Information Management for z/OS NetView Bridge Adapter

The Tivoli Information Management for z/OS NetView Bridge Adapter provides you with a sample startup JCL procedure. This section explains how to use this procedure to get the NetView Bridge adapter working using these steps:

- Assign an application identifier (APPLID) for the server address space and a unique program-to-program interface (PPI) receiver queue name (RCVQUEUE) in the JCL procedure's EXEC statement as described in "The EXEC Statement" on page 9.
- Create the data set INFOBRDS as described in "The User-Supplied Input Data Set" on page 10.
- Create following output data sets as described in "The User-Supplied Output Data Sets" on page 17:
  - IBRPRINT
  - HLAPILOG
  - APIPRINT
  - SYSPRINT
  - SYSUDUMP

The example on page 8 shows a typical JCL startup procedure for a Tivoli Information Management for z/OS NetView Bridge Adapter server address space. You must modify the HCL EXEC supplied with the adapter to fit your particular user environment. An explanation of each part of the procedure shown in boldface type follows the example.

In the JCL example on page 8, the values for the procedure name, APPLID, and receiver queue name must also be unique for each adapter server address space. You can use the JCL to start multiple copies of the adapter if you modify it to contain these unique parameters each time. See "Creating Additional Copies of a Server" on page 151 for more information.

## Allocating the Checkpoint File VSAM Data Set

A sample JCL job, EYLSJ008, is provided with AutoBridge to define VSAM clusters for the AutoBridge checkpoint files. Table 27 lists the statements in EYLSJ008, their purpose, and the members that contain their VSAM cluster information.

*Table 27. VSAM data for checkpoint file allocation*

| Statement in EYLSJ008 | Purpose | Member |
|---|---|---|
| //STEP1 EXEC | Delete existing VSAM checkpoint file | EYLSID01 |
| //STEP2 EXEC | Allocate VSAM checkpoint file | EYLSI101 |

Follow these steps to define the clusters using EYLSJ008:

1. Before running EYLSJ008, review the members listed in Table 27 on page 150 for VSAM (cluster) data set names or VOL(*xxxxxx*) changes.
2. Update the EYLSJ008 JCL to reflect the correct DASD type, data set names, and any other information that is unique to your environment.

   **Note:** For better performance, place the VSAM database INDEX and DATA components on different devices.
3. Run EYLSJ008.
4. After you run EYLSJ008 the first time, remove the asterisk (*) that follows the slashes (//) in the //STEP1 EXEC statement of the job. The //STEP1 EXEC statement deletes the previously allocated data sets.
5. When EYLSJ008 has finished processing, add the following statement to the NetView startup procedure to define the VSAM DDNAME:
   ```
   EYLVSAM    DD   DSN=name_specified_in_JCL,
                   DISP=SHR,AMP=AMORG
   ```

## Customizing the DSIPARM DSIDMN Member

AutoBridge contains two data services tasks (DSTs). To define these DSTs, copy the SEYLSPL member EYLDMN into the DSIDMN member of DSIPARM. Copy the two SEYLSPL DST definition members, EYLATMEM and EYLATSUB, into a data set of the DSIPARM concatenation list.

Your Tivoli Information Management for z/OS administrator must add the application identifier (APPLID) definition to the MASTER class record for Tivoli Information Management for z/OS.

## Creating Additional Copies of a Server

Each transaction request from NetView is queued in the bridge dispatcher until the Tivoli Information Management for z/OS NetView Bridge Adapter server becomes available. A high volume of transaction traffic may cause a bottleneck in your system. At these times, you may want to create more than one copy of an adapter server. Multiple copies provide greater throughput for your transactions by giving the dispatcher more than one server in which to send the queued requests.

Keep in mind, however, that each copy of a server costs you in terms of storage and possible degradation of performance of both NetView and Tivoli Information Management for z/OS.

To create an additional copy of an existing NetView Bridge adapter server, make a separate copy of the JCL procedure and modify the copy as follows:

1. Assign a unique receiver PPI queue name for each additional server.
2. Define a unique APPLID for each extra server.

   **Note:** The application identifier must be a unique ID defined to Tivoli Information Management for z/OS and included in the INITCLAS privilege class. If you use a security product, such as RACF®, you may also want to define this ID to that security product. For additional security considerations, see the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*.
3. Define unique output data sets for each server's use.

**Note:** If you want to run a second copy of the adapter with different initialization parameters, you can modify the INFOBRDS data set or create a new copy of the data set and point to it in the modified JCL procedure. If you want to run two copies of the adapter with the same parameters, both copies can use the same INFOBRDS data set.

You do not need to change any programming of NetView commands or NetView Bridge definitions to use the additional copies of a server.

## Customizing the PIDT, PIPT, and Alias Tables

Use the information in the *Tivoli Information Management for z/OS Application Program Interface Guide* to customize the PIDT, PIPT, and alias tables as necessary. All values of the server can share the PIDT, PIPT, and Alias Tables

While modifying these tables for AutoBridge, you may want to make the changes necessary for implementing the PostProcessor. To implement the PostProcessor, you must define the PostProcessor Marker structured word (S7C00) in the PIDT and, optionally, in the alias table. The examples contained in “Modifying AutoBridge’s Tivoli Information Management for z/OS Interface” on page 179 illustrate the necessary modifications. If BYPASS_PANEL_PROCESSING=YES is specified in the INFOBRDS data set, then the values specified for IBCREATE, IBUPDATE, IBSEARCH, and INQVIEW will be used as data view names. Even if BYPASS_PANEL_PROCESSING=NO is specified in the INFOBRDS data set, if USE_DATA_VIEW=YES is specified, these values will be used as data view names.

## Customizing the NetView Automation Table

You may need to edit your NetView automation table to add the automation steps necessary for implementing AutoBridge. Your changes take effect the next time the NetView automation table load command is issued (AUTOTBL MEMBER=*xxxxxxxx*).

The following are suggested NetView automation table additions for AutoBridge:

- If you want AutoBridge to start automatically during NetView startup, make an entry to trap a known message and issue the ABRIDGE command. The following is an example of this type of entry:

```
***********************************************************************
*    NEXT STATEMENT NEEDED TO INVOKE THE AUTOBRIDGE APPLICATION       *
*    ONCE THE SSI TASK HAS INITIALIZED                                *
***********************************************************************
 IF MSGID='DSI530I' & TEXT=.'CNMCSSIR'. & DOMAINID=%MYDOMAIN%
   THEN EXEC(CMD('ABRIDGE') ROUTE(ONE AUTO1));
*
```

- To allow AutoBridge messages (prefixed with EYL) to be displayed on ABMENU operator screens, make an entry in the NetView automation table to route these messages to operators in the AutoBridge group. (Whenever you display any ABMENU panel, you are automatically added to the AutoBridge group.) This entry will drive the AutoBridge command list EYLESUPD, which creates a common global value for the AutoBridge message text used by the VIEW processor. The following is an example of this type of entry:

```
***********************************************************************
IF MSGID = 'EYL' .
   & TEXT = MESSAGE
   THEN EXEC(CMD('EYLESUPD ' MESSAGE) ROUTE (ALL +EYL))
   NETLOG(Y) DISPLAY(Y) SYSLOG(Y) CONTINUE(Y);
*
```

On a resident NetView, you may also choose to have PostProcessor messages (prefixed with EYM) be displayed on the operator panel. The following is an example of this type of entry:

```
**********************************************************************
IF MSGID = 'EYM' .
   & TEXT = MESSAGE
   THEN EXEC(CMD('EYLESUPD ' MESSAGE) ROUTE (ALL +EYL))
   NETLOG(Y) DISPLAY(Y) SYSLOG(Y) CONTINUE(Y);
*
```

- Edit the NetView automation table to drive the AutoBridge API for any MSUs and messages requiring AutoBridge processing. See "NetView AutoBridge Implementation Scenarios" on page 129 for examples of these kinds of entries.

### Creating the Initialization Table

The initialization table resides in the EYLATINT member of the SEYLSPL data set. Make any changes necessary and copy to DSIPARM.

Use the sample member EYLATINT and the data on the initialization table worksheets on page 231 to code an initialization table for each NetView.

### Creating the Process Table for Resident NetView

The process table resides in the EYLATPRO member of the SEYLSPL data set. Process table segments must reside either in the EYLATPRO member of DSIPARM or in another member of DSIPARM that is referred to from the process table by an %INCLUDE statement.

Use the sample member EYLATPRO and the data on the process table worksheet on page 233 to code the process table for your AutoBridge installation.

### Creating the Mapping Table for Resident NetView

The mapping table resides in the EYLATMAP member of the SEYLSPL data set. Mapping table segments must reside either in the EYLATMAP member of DSIPARM or in another member of DSIPARM that is referred to from the mapping table by an %INCLUDE statement.

Use the sample member EYLATMAP and the data on the mapping table worksheet on page 234 to code the mapping table for your AutoBridge installation.

### Creating the Filter Table for Resident NetView

The filter table resides in the EYLATFIL member of the SEYLSPL data set.

Use the sample member EYLATFIL and the data on the filter table worksheet on page 235 to code the filter table for your AutoBridge installation. Then copy it to the DSIPARM data set.

## Setting Up Remote NetViews

There are additional things to consider when implementing a remote NetView. Notice in the following diagram that on the remote NetView the NetView Bridge API sends requests to the resident's bridge dispatcher (via the resident's remote dispatcher). The bridge dispatcher sends replies to the remote's remote dispatcher. Thus, each NetView needs its own remote dispatcher.

You don’t specify the remote dispatcher in your AutoBridge call. Your automation table calls could be identical on the resident and all remotes (if you have the same process table), because the dispatcher parameter is the name of the bridge dispatcher on the resident NetView.

```
ABAPI PROCESS_SEGMENT BRGDHIPR MSUSEG
```

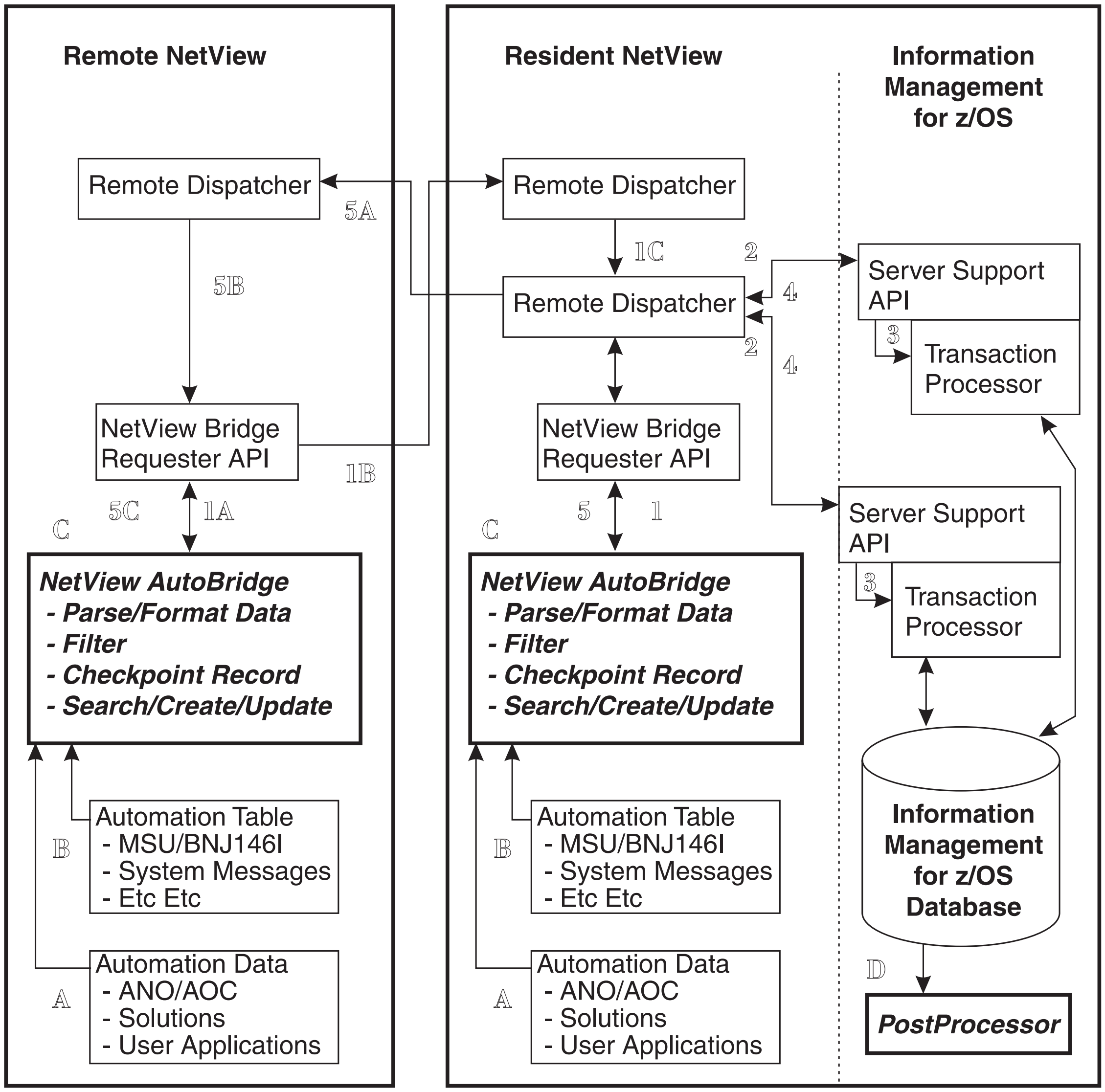

Remote NetView
Resident NetView
Information Management for z/OS
Remote Dispatcher
5A
5B
Remote Dispatcher
1C
2
Remote Dispatcher
4
2
4
Server Support API
3
Transaction Processor
NetView Bridge Requester API
1B
NetView Bridge Requester API
5C
1A
C
5
1
C
Server Support API
3
Transaction Processor
NetView AutoBridge - Parse/Format Data - Filter - Checkpoint Record - Search/Create/Update
NetView AutoBridge - Parse/Format Data - Filter - Checkpoint Record - Search/Create/Update
Information Management for z/OS Database
B
Automation Table - MSU/BNJ146I - System Messages - Etc Etc
B
Automation Table - MSU/BNJ146I - System Messages - Etc Etc
D
PostProcessor
A
Automation Data - ANO/AOC - Solutions - User Applications
A
Automation Data - ANO/AOC - Solutions - User Applications

Figure 12. Interaction of Remote and Bridge Dispatchers

Follow these steps to set up NetView Bridge and AutoBridge components on each separate remote NetView:

Table 28. Remote NetView setup checklist

| ✓ | Step |
|---|---|
| | Add operator IDs for the remote bridge dispatcher and checkpoint manager tasks on the remote NetViews as described in “Adding an Operator ID for the Dispatcher Autotask” on page 155. |

*Table 28. Remote NetView setup checklist (continued)*

| ✓ | Step |
|---|---|
| | Create autotask profiles for the new operator IDs on the remote NetViews as described in “Creating NetView Bridge Dispatcher Profiles” on page 156. |
| | Add NetView Bridge and AutoBridge command model statements to the remote NetViews as described in “Adding NetView Bridge Command Model Statements to NetView” on page 156. |
| | Modify the NetView procedure JCL to include command list, panel, message, and STEPLIB data sets as described in “Modifying the NetView Procedure JCL” on page 156. |
| | Allocate a VSAM data set for the AutoBridge checkpoint files as described in “Allocating the Checkpoint File VSAM Data Set” on page 157. |
| | Customize the DSICMD and DSIDMN members of the concatenated DSIPARM data set as described in “Customizing the DSIPARM DSIDMN Member” on page 157. |
| | Customize the NetView automation table as described in “Customizing the NetView Automation Table” on page 157. |
| | Customize the initialization table as described in “Creating the Initialization Table for a Remote NetView” on page 158. |
| | Create the process table as described in “Creating the Process Table for a Remote NetView” on page 159. |
| | Create the mapping table as described in “Creating the Mapping Table for a Remote NetView” on page 159. |
| | Create the filter table as described in “Creating the Filter Table for a Remote NetView” on page 159. |

## Adding an Operator ID for the Dispatcher Autotask

You must define at least one operator ID on each remote NetView for each of the following autotasks:

- Remote dispatcher
- Checkpoint managers—one for each NetView Bridge dispatcher on the resident NetView that will receive transactions from this NetView

The operator IDs you define will serve as the NetView interface for these autotasks. The NetView Bridge Requester API uses these operator IDs to request the associated autotasks.

To define the IDs, you must modify the DSIOPF member of DSIPARM. The member EYLREM in the sample data set SEYLSPL contains the following operator ID definitions that you can copy into DSIOPF and modify as needed:

```
REMOPERA     OPERATOR    PASSWORD=REMOPERA
             PROFILEN    EYLPRFRD
EYLTCHK1     OPERATOR    PASSWORD=EYLTCHK1
             PROFILEN    EYLPRFAO
EYLTCHK2     OPERATOR    PASSWORD=EYLTCHK2
             PROFILEN    EYLPRFAO
```

Note that each definition in this example contains a pointer to an associated profile. The following section describes how to create these profiles.

Your ID definitions take effect the next time NetView is recycled.

## Creating NetView Bridge Dispatcher Profiles

You must create new members in the NetView DSIPRF data set having the profile names referenced in the operator ID definitions you created in the previous section. The profile specifies the NetView Bridge command REMOTEBR to initialize remote access to the bridge.

By customizing the sample profiles provided with AutoBridge (EYLPRFAO and EYLPRFRD), you can create profiles for the checkpoint manager tasks and a remote dispatcher.

The following are examples of PROFILE statements that correspond to the example operator IDs in the previous section:

```
  EYLPRFRD   PROFILE   IC=REMOTEBR
             AUTH   MSGRECVR=NO,CTL=GLOBAL
             OPCLASS 1,2,6
             END
```

For information on the REMOTEBR command, see “REMOTEBR Command” on page 147.

## Adding NetView Bridge Command Model Statements to NetView

You must define the NetView Bridge commands to NetView by using command model (CMDMDL) statements. These statements are in the DSICMD member of the DSIPARM data set.

To define these statements for the remote NetView Bridge, remove the asterisks that comment out the following command model statements in the DSICMD member:

```
TRANRCV   CMDMDL MOD=DSINBRCV,TYPE=R,RES=Y
TRANSND   CMDMDL MOD=DSINBSND,TYPE=R,RES=Y
DSINBR62  CMDMDL MOD=DSINBR62,TYPE=R,PARSE=N,RES=Y
REMOTEBR  CMDMDL MOD=DSINBREM,TYPE=R,PARSE=Y,RES=Y
DSINBRLG  CMDMDL MOD=DSINBRLG,TYPE=R,PARSE=Y,RES=Y
```

To define command model statements for AutoBridge, copy the member EYLCMD from the SEYLSPL data set into the DSICMD member.

## Modifying the NetView Procedure JCL

Modify the NetView startup procedure JCL to include the AutoBridge data sets. You must add these statements to the command list, panel, message, and STEPLIB data set concatenation lists. If you have changed the data set allocation statements or renamed the data sets after they were unloaded, use the new names that you have chosen.

**Command lists**

Your command list data set concatenation list should contain the following lines:

```
//DSICLD DD DSN=current clists
//       DD DSN=EYL.SEYLCL1
//       DD DSN=EYL.SEYLSPL
```

**Panels**

Your panel data set concatenation list should contain the following lines:

```
//CNMPNL1 DD DSN=current panels
//        DD DSN=EYL.SEYLPN1,DISP=SHR
//        DD DSN=EYL.SEYLPN2,DISP=SHR
```

**Messages**

Your message data set concatenation list should contain the following lines:

```
//DSIMSG DD DSN=current messages
//       DD DSN=EYL.SEYLMSG,DISP=SHR
```

**STEPLIB**

Your STEPLIB data set concatenation list should contain the following lines:

```
//STEPLIB DD DSN=current steplibs
//        DD DSN=EYL.SEYLMOD
```

**Note:** All libraries in this STEPLIB must be authorized.

## Allocating the Checkpoint File VSAM Data Set

A sample JCL job, EYLSJ008, is provided with AutoBridge to define VSAM clusters for the AutoBridge checkpoint files. Table 29 lists the statements in EYLSJ008, their purpose, and the members that contain their VSAM cluster information.

*Table 29. VSAM data for checkpoint file allocation*

| Statement in EYLSJ008 | Purpose | Member |
|---|---|---|
| //STEP1 EXEC | Delete existing VSAM checkpoint file | EYLSID01 |
| //STEP2 EXEC | Allocate VSAM checkpoint file | EYLSI101 |

Follow these steps to define the clusters using EYLSJ008:

1. Before running EYLSJ008, review the members listed in Table 29 for VSAM (cluster) data set names or VOL(*xxxxxx*) changes.
2. Update the EYLSJ008 JCL to reflect the correct DASD type, data set names, and any other information that is unique to your environment.

   **Note:** For better performance, place the VSAM database INDEX and DATA components on different devices.
3. Run EYLSJ008.
4. After you run EYLSJ008 the first time, remove the asterisk (*) that follows the slashes (//) in the //STEP1 EXEC statement of the job. The //STEP1 EXEC statement deletes the previously allocated data sets.
5. When EYLSJ008 has finished processing, add the following statement to the NetView startup procedure to define the VSAM DDNAME:

   ```
   EYLVSAM    DD   DSN=name_specified_in_JCL,
                   DISP=SHR,AMP=AMORG
   ```

## Customizing the DSIPARM DSIDMN Member

AutoBridge contains two data services tasks (DSTs). To define these DSTs, copy the SEYLSPL member EYLDMN into the DSIDMN member of DSIPARM. Copy the two SEYLSPL DST definition members, EYLATMEM and EYLATSUB, into a data set of the DSIPARM concatenation list.

## Customizing the NetView Automation Table

You may need to customize your NetView automation table for use with AutoBridge. Edit your NetView automation table to add the automation steps necessary for implementing AutoBridge. Your changes take effect the next time the NetView automation table load command (AUTOTBL MEMBER=*xxxxxxxx*) is issued.

The following are suggested NetView automation table additions for AutoBridge:

- If you want AutoBridge to start automatically during NetView startup, make an entry to trap a known message and issue the ABRIDGE command. The following is an example of this type of entry:

```
***********************************************************************
*    NEXT STATEMENT NEEDED TO INVOKE THE AUTOBRIDGE APPLICATION       *
*    ONCE THE SSI TASK HAS INITIALIZED                                *
***********************************************************************
 IF MSGID='DSI530I' & TEXT=.'CNMCSSIR'. & DOMAINID=%MYDOMAIN%
   THEN EXEC(CMD('ABRIDGE') ROUTE(ONE AUTO1));
*
```

- To allow AutoBridge messages (prefixed with EYL) to be displayed on ABMENU operator screens, make an entry in the NetView automation table to route these messages to operators in the AutoBridge group. (Whenever you display any ABMENU screen, you are automatically added to the AutoBridge group.) This entry will drive the AutoBridge command list EYLESUPD, which creates a common global value for the AutoBridge message text used by the VIEW processor. The following is an example of this type of entry:

```
***********************************************************************
IF MSGID = 'EYL' .
   & TEXT = MESSAGE
   THEN EXEC(CMD('EYLESUPD ' MESSAGE) ROUTE (ALL +EYL))
   NETLOG(Y) DISPLAY(Y) SYSLOG(Y) CONTINUE(Y);
*
```

  You may also choose to have PostProcessor messages (prefixed with EYM) be displayed on the operator panel. The following is an example of this type of entry:

```
***********************************************************************
IF MSGID = 'EYM' .
   & TEXT = MESSAGE
   THEN EXEC(CMD('EYLESUPD ' MESSAGE) ROUTE (ALL +EYL))
   NETLOG(Y) DISPLAY(Y) SYSLOG(Y) CONTINUE(Y);
*
```

- Edit the NetView automation table to drive the AutoBridge API for any MSUs and messages requiring AutoBridge processing. See “NetView AutoBridge Implementation Scenarios” on page 129 for examples of these kinds of entries.

## Creating the Initialization Table for a Remote NetView

The initialization table resides in the EYLATINT member of the DSIPARM data set. Make any changes necessary and copy to the DSIPARM data set.

Use the sample member EYLINTRM in SEYLSPL and the data on the initialization table worksheets on page 231 to create the initialization table, EYLATINT, for each remote NetView. The EYLATINT table is different on the remote NetViews. The most obvious difference is

```
 AUTOBRIDGE = REMOTE
```

There is also the Dispatcher/Checkpoint task definitions difference. The remote NetView has only a remote dispatcher and checkpoint task(s). The bridge dispatcher names are defined only to associate a dispatcher with the sender’s checkpoint task.

```
*-- Remote dispatcher on this NetView ---------------------------------
RDISPATCH = BRGREMOP

*-- Bridge dispatcher, checkpoint mgr and adapters segment -------------
BEGIN DISPATCHER BRGDHIPR <-- This exists on the resident only
```

```
CHECKPT = EYLTCHKH
END DISPATCHER BRGDHIPR

BEGIN DISPATCHER BRGDLOPR <-- This exists on the resident only
CHECKPT = EYLTCHKL
END DISPATCHER BRGDLOPR
```

The remote bridge autotask BRGREMOP and the checkpoint tasks EYLTCHKH and EYLTCHKL must exist on this remote NetView. The bridge dispatcher tasks BRGDHIPR and BRGDLOPR exist on the resident NetView only.

On this remote NetView a call is made to AutoBridge as:

```
 ABAPI PROCESS_SEGMENT BRGDHIPR MSUSEG
```

The record is created and the checkpoint manager running on task EYLTCHKH forwards this transaction via the high performance transport (LU6.2 session) to the bridge dispatcher BRGDHIPR on the resident NetView. This is processed by the bridge adapter (also known as database server) associated with this bridge dispatcher. The results are sent to the remote dispatcher BRGREMOP on the remote NetView and forwarded to the originator EYLTCHKH.

### Creating the Process Table for a Remote NetView

The process table resides in the EYLATPRO member of the DSIPARM data set. Process table segments must reside either in the EYLATPRO member or in another member of DSIPARM that is referred to from the process table by an %INCLUDE statement.

Use the sample member EYLATPRO in SEYLSPL and the data on the process table worksheet on page 233 to code the process table for your AutoBridge installation.

### Creating the Mapping Table for a Remote NetView

The mapping table resides in the EYLATMAP member of the DSIPARM data set. Mapping table segments must reside either in the EYLATMAP member or in another member of DSIPARM that is referred to from the mapping table by an %INCLUDE statement.

Use the sample member EYLATMAP in SEYLSPL and the data on the mapping table worksheet on page 234 to code the mapping table for your AutoBridge installation.

### Creating the Filter Table for a Remote NetView

The filter table resides in the EYLATFIL member of the DSIPARM data set.

Use the sample member EYLATFIL in SEYLSPL and the data on the filter table worksheet on page 235 to code the filter table for your AutoBridge installation.

## VTAM List

Ensure that the VTAM List specification for the resident and all remote NetViews include the APPC=YES parameter to allow LU6.2 sessions. For example:

```
'SYS1.VTAMLST(CNM0100)'
***********************************************************************
* IF LU 6.2 OR GRAPHIC MONITOR FACILITY IS NOT BEING USED,           *
* THE APPC KEYWORD CAN BE REMOVED FROM NETVIEW'S APPL                *
* STATEMENT.                                                         *
***********************************************************************
CNM01 APPL AUTH=(NVPACE,ACQ,PASS),PRTCT=CNM01,EAS=6,                  X
           MODETAB=AMODETAB,DLOGMOD=DSILGMOD,APPC=YES
```

## Required NetView Tasks

When using Remote NetViews, list these tasks and make sure that they are started:

```
LIST DSIHPDST
LIST DSI6DST
LIST DSIUDST
```

These would normally be started when you start NetView in the CNMSTART clist (CNME1015).

## Using RMTCMD to Test Connectivity

To verify that two NetViews can communicate via an LU6.2 session, you can use the RMTCMD. To do this, issue a RMTCMD to the other NetView specifying an autotask name. The autotask doesn't have to be the checkpoint task; any autotask defined at the target is acceptable. But, if you use the checkpoint task name, it will also check that the task is defined correctly.

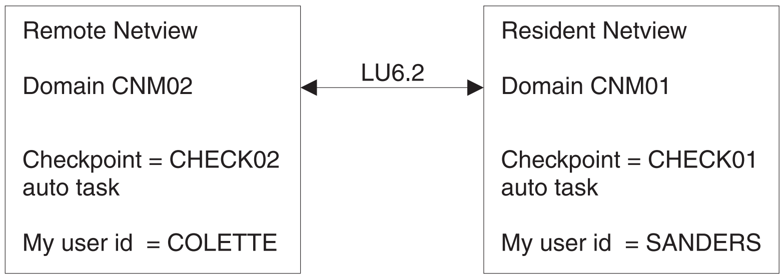


*Figure 13. Remote and the resident NetViews*

In most cases the checkpoint name and "my user ID" would be identical on the two NetViews. Different names are shown here so that it's easier to follow this example.

- On the resident NetView:

  ```
  you enter --> RMTCMD LU=CNM02,OPERID=CHECK02,MSG COLETTE,OK FROM CNM01
  you see --> DSI001I MESSAGE SENT TO COLETTE
  ```

- On the remote NetView:

  ```
  you see --> DSI039I MSG FROM CHECK02 : OK FROM CNM01
  ```

- On the remote NetView:

  ```
  you enter --> RMTCMD LU=CNM01,OPERID=CHECK01,MSG SANDERS,OK FROM CNM02
  you see --> DSI001I MESSAGE SENT TO SANDERS
  ```

- On the remote NetView:

  ```
  you see --> DSI039I MSG FROM CHECK01 : OK FROM CNM02
  ```

If the RMTCMD fails, messages will be logged reporting the problem.

```
DSI072A RMTCMD COMMAND ABORTED. DSIHSNDS RETURN CODE = 220
```

This indicates that the DSIHSNDS macro (Send High Performance Message Unit) received rc = 220. *NetView Customization: Using Assembler* documents that a return code of 220 means ″DSIHPDST task inactive″. Start it.

Another failure message might be something like:

```
DSI769I CNM01 RPL_EXIT APPCCMD CONTROL = SEND FAILED - REG15 = X'00', REG
DWO570I UNABLE TO ESTABLISH REMOTE SESSION ON USIBMTA.CNM02 WITH SENSE: X
DWO575I RMTCMD TERMINATED ON USIBMTA.CNM02 WITH SENSE: X'08A80003'
```

You can enter **SENSE 08A80003** to get an explanation of the sense code.

# 13 Using the NetView AutoBridge PostProcessor

This chapter describes the AutoBridge PostProcessor facility, its function, requirements, and operation, as well as how to install and implement it. The following sections provide this information:

- "PostProcessor Overview" describes what the PostProcessor does and why you might want to use it.
- "Installing the PostProcessor" on page 167 describes the steps necessary to install the PostProcessor.
- "Setting Up the PostProcessor" on page 173 describes the setup tasks for implementing the PostProcessor.
- "Running the PostProcessor" on page 181 discusses considerations of operating the PostProcessor.
- "Creating a Mapping Reference Record" on page 174 describes the mapping reference records that the PostProcessor uses to modify AutoBridge-created transactions.
- "AutoBridge PostProcessor User Exits" on page 184 describes the AutoBridge PostProcessor user exits. The function, overview, description, input, output, and return codes are provided for each user exit.

## PostProcessor Overview

AutoBridge's *PostProcessor* facility locates and processes records in the Tivoli Information Management for z/OS database. The PostProcessor accesses AutoBridge-created records and modifies them according to a set of instructions that you supply. The PostProcessor's flexible design enables it to process records created through your customized Tivoli Information Management for z/OS panels.

Figure 5 on page 80 shows a representation of how the PostProcessor supplements the Tivoli Information Management for z/OS record data created by AutoBridge.

### When to Use the PostProcessor

You should use the PostProcessor if you need to supplement the records created by AutoBridge with additional data to complete each record as though it had been entered on a terminal by an operator. Because AutoBridge uses the Tivoli Information Management for z/OS Application Program Interface (API), which bypasses the panels that allow operators to add record data, some of this data may be missing from records entered by AutoBridge. The panels AutoBridge bypasses may invoke routines and program exits that generate new data to add to the record or copy it from some other source.

Consult with your Tivoli Information Management for z/OS administrator to determine if your Problem Management panels invoke control panels, program exits, or terminal simulator panels (TSPs) that modify Tivoli Information Management for z/OS records. If so, you will need to use the PostProcessor to ensure that AutoBridge records are equivalent in content to records entered manually by Tivoli Information Management for z/OS operators.

## PostProcessor Function

The PostProcessor performs two major functions. First, it initializes the PostProcessor environment and detects the creation of records by the Tivoli Information Management for z/OS API. Second, it performs the actual processing of these records.

For optimum performance, the PostProcessor resides on your system as a set of background tasks, one for each Tivoli Information Management for z/OS NetView Bridge Adapter. Since AutoBridge supports up to four NetView Bridge dispatchers on the resident NetView, and each dispatcher can have up to four associated adapters, there can be as many as 16 PostProcessor tasks on the resident NetView.

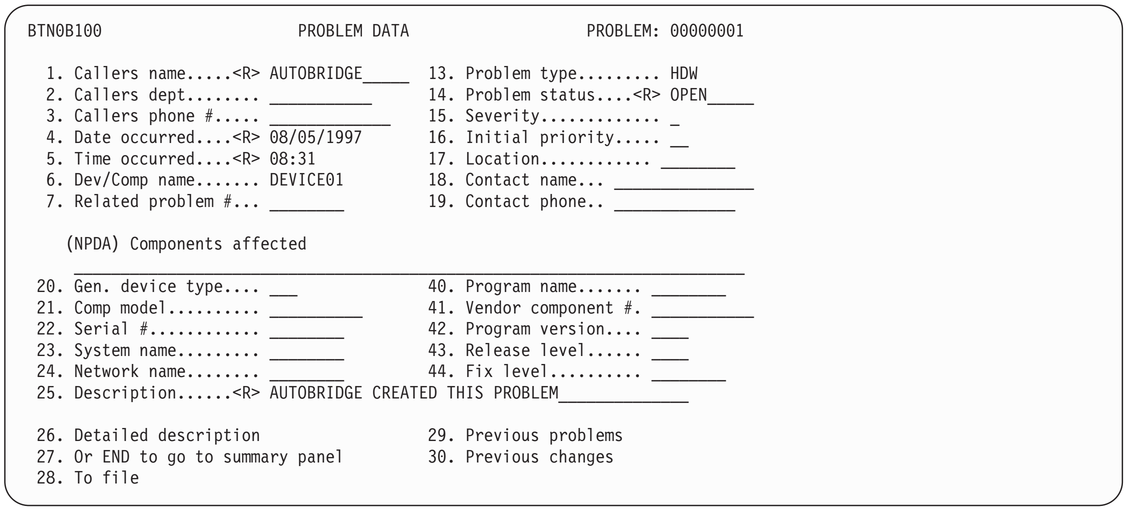
A PostProcessor task is activated when an AutoBridge transaction occurs on its associated adapter. Once an individual PostProcessor task is activated, it searches the database for AutoBridge-created records that have not yet been post-processed. (AutoBridge-created records contain a unique structured word entry that distinguishes them from other records in the database.) It then processes each record in the search results list according to instructions that you specify in a new type of Tivoli Information Management for z/OS record, provided with AutoBridge, called a *mapping reference record*. The mapping reference record contains entries that correspond to those items on your Tivoli Information Management for z/OS panels that invoke processes to modify or generate record data.

The PostProcessor handles records in a way that prevents post-processing errors from corrupting Tivoli Information Management for z/OS data. The PostProcessor task makes a copy of the AutoBridge-created record, which it then processes as specified in the mapping reference record. After this copy is successfully processed and filed, the original record is deleted. The PostProcessor then issues message EYM105I to indicate that record *new_record_id* has been post-processed and record *old_record_id* has been deleted.

## PostProcessor Example

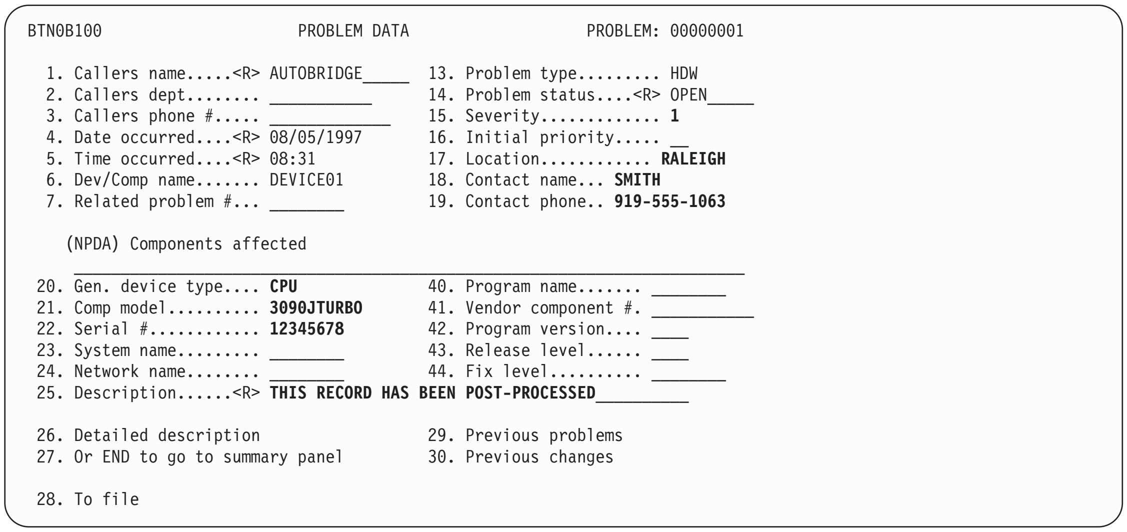
As an example of PostProcessor operation, consider a record that is displayed on an *Integration Facility* panel as shown in this panel.

**Note:** Although this example uses Integration Facility panels, the PostProcessor operates the same way with base Tivoli Information Management for z/OS panels.



```
BTN0B100                   PROBLEM DATA                 PROBLEM: 00000001

  1. Callers name.....<R> AUTOBRIDGE_____   13. Problem type......... HDW
  2. Callers dept........ ___________       14. Problem status....<R> OPEN_____
  3. Callers phone #..... _____________     15. Severity............. _
  4. Date occurred....<R> 08/05/1997        16. Initial priority..... __
  5. Time occurred....<R> 08:31             17. Location............ ________
  6. Dev/Comp name....... DEVICE01          18. Contact name... _______________
  7. Related problem #... ________          19. Contact phone.. _____________

    (NPDA) Components affected
   ________________________________________________________________________
 20. Gen. device type.... ___               40. Program name....... ________
 21. Comp model.......... __________        41. Vendor component #. ___________
 22. Serial #............ ________          42. Program version.... ____
 23. System name......... ________          43. Release level...... ____
 24. Network name........ ________          44. Fix level.......... ________
 25. Description......<R> AUTOBRIDGE CREATED THIS PROBLEM______________

 26. Detailed description                   29. Previous problems
 27. Or END to go to summary panel          30. Previous changes
 28. To file
```

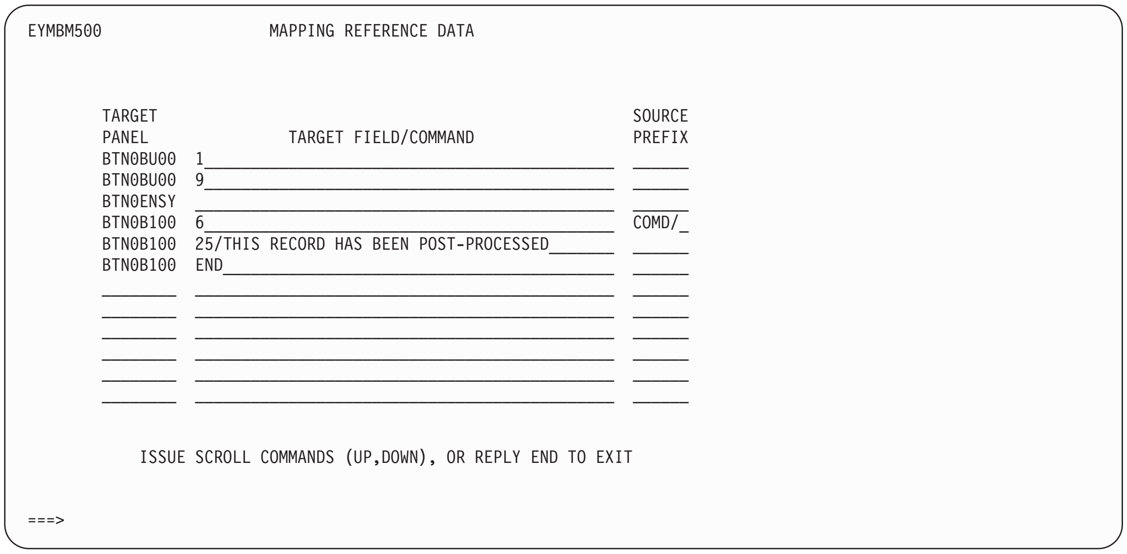
In the next panel, problem record 00000001 is created by AutoBridge with basic problem data. Had an operator entered the same data through the Integration Facility panels, the PROBLEM DATA panel would contain more data as shown in this example.



```
BTN0B100                   PROBLEM DATA                 PROBLEM: 00000001

  1. Callers name.....<R> AUTOBRIDGE_____   13. Problem type......... HDW
  2. Callers dept........ ___________       14. Problem status....<R> OPEN_____
  3. Callers phone #..... _____________     15. Severity............. 1
  4. Date occurred....<R> 08/05/1997        16. Initial priority..... __
  5. Time occurred....<R> 08:31             17. Location............ RALEIGH
  6. Dev/Comp name....... DEVICE01          18. Contact name... SMITH
  7. Related problem #... ________          19. Contact phone.. 919-555-1063

    (NPDA) Components affected
   ________________________________________________________________________
 20. Gen. device type.... CPU               40. Program name....... ________
 21. Comp model.......... 3090JTURBO        41. Vendor component #. ___________
 22. Serial #............ 12345678          42. Program version.... ____
 23. System name......... ________          43. Release level...... ____
 24. Network name........ ________          44. Fix level.......... ________
 25. Description......<R> THIS RECORD HAS BEEN POST-PROCESSED__________

 26. Detailed description                   29. Previous problems
 27. Or END to go to summary panel          30. Previous changes

 28. To file
```

Note the additional data shown in the previous panel. This data is generated when a value is entered into field 6 (Dev/Comp name). Data entered in this field triggers a call to a program exit that copies data into the problem record from the record for hardware component DEVICE01. The AutoBridge-created record does not include this additional data because it is created through the Tivoli Information Management for z/OS API, which cannot access this panel to trigger the logic behind field 6.

The PostProcessor, however, is able to reproduce the action that triggers the logic behind field 6. To do this, the PostProcessor locates the value for field 6 in the record, DEVICE01, then builds a command to enter the value into that field. The PostProcessor is instructed to do so by the mapping reference record fields that correspond to this panel. For the PostProcessor to accomplish the processing required for this example, the mapping reference record would appear as shown in the next panel.



```
EYMBM500                MAPPING REFERENCE DATA



       TARGET                                               SOURCE
       PANEL             TARGET FIELD/COMMAND               PREFIX
       BTN0BU00  1_______________________________________   ______
       BTN0BU00  9_______________________________________   ______
       BTN0ENSY  ________________________________________   ______
       BTN0B100  6_______________________________________   COMD/_
       BTN0B100  25/THIS RECORD HAS BEEN POST-PROCESSED______   ______
       BTN0B100  END_____________________________________   ______
       ________  ________________________________________   ______
       ________  ________________________________________   ______
       ________  ________________________________________   ______
       ________  ________________________________________   ______
       ________  ________________________________________   ______
       ________  ________________________________________   ______


           ISSUE SCROLL COMMANDS (UP,DOWN), OR REPLY END TO EXIT


===>
```

The PostProcessor would access and process the data shown in the Mapping Reference panel on page 166 as follows:

1. AutoBridge creates a record, activating the PostProcessor when it invokes the Tivoli Information Management for z/OS API.
2. The PostProcessor searches for all records that were created by AutoBridge and compiles these records in a search results list for processing. This example describes the processing of the first record only.
3. The PostProcessor selects the record for update to prevent the record from being modified by another operator while it is being post-processed. The PostProcessor locates the structured word in the record that specifies the name of the mapping reference record to be used. It then reads this mapping reference record to determine the post-processing sequence.
4. The PostProcessor makes a copy of the record and begins processing it using the information in the mapping reference record. In the Integration Facility, the copy of the record is processed starting on the Problem Summary panel, BTN0BU00.
5. The PostProcessor reads the name of the currently displayed panel (BTN0BU00) and locates the first row in the mapping reference record having a matching Target Panel value. This is the first row in the Mapping Reference panel on page 166. Its Target Field/Command field contains the value `1`.

   This row instructs the PostProcessor to enter the value `1` on the command line, which initiates data entry on the Problem Summary panel. The PostProcessor marks the row in the mapping reference record as having been processed, so that it will not be accessed again for this record. The Integration Facility dialog then flows from panel BTN0BU00 to panel BTN0B100, the Problem Data panel shown on page 165.
6. The PostProcessor looks for the first available row in the mapping reference record having a Target Panel value of BTN0B100. This is the fourth row in the Mapping Reference panel on page 166. It has a Target Field/Command value of `6` and a source prefix value of `COMD/`.

   This row instructs the PostProcessor to locate the record data associated with the prefix word COMD/ and enter it into field 6. (You can determine the prefix word for a value by using the Tivoli Information Management for z/OS VIEW INTERNALS command.).

This value is DEVICE01, so the PostProcessor issues the command `6,DEVICE01`, and then marks the row in the mapping reference record, making it unavailable to subsequent searches. With this command processed, the Integration Facility dialog flows to a control panel that calls a program exit to copy the data from the Hardware Component field. Control then returns to panel BTN0B100.

7. The PostProcessor looks for the first available row in the mapping reference record having a Target Panel value of BTN0B100. This is the fifth row in the Mapping Reference panel on page 166. Its Target Field/Command contains the value `25/THIS RECORD HAS BEEN POSTPROCESSED`, which causes a message to be placed in the Description field, number 25. The PostProcessor then marks the row as having been processed.

   **Note:** In the mapping reference record, slashes (/) are used as delimiters (rather than the commas that are used on Tivoli Information Management for z/OS panels).

8. The PostProcessor looks for the first available row in the mapping reference record having a Target Panel value of BTN0B100—the sixth row in the Mapping Reference panel on page 166. The Target Field/Command field in this row contains the value `END`, which instructs the PostProcessor to enter the value `END` on the command line. The PostProcessor then marks the row as having been processed. With this command processed, the panel flow returns to panel BTN0BU00.

9. The PostProcessor looks for the first available row in the mapping reference record having a Target Panel value of BTN0BU00. This is the second row in the list in the Mapping Reference panel on page 166, with a Target Field/Command value of `9`. This row instructs the PostProcessor to enter the value `9` on the command line. On the Problem Summary panel, this is the command to file the record. The PostProcessor then marks the row as having been processed.

   As a result of this command, Tivoli Information Management for z/OS files the record and the flow returns to primary options panel, BTN0ENSY (assuming a systems administrator privilege class in the Integration Facility).

10. The PostProcessor looks for the first available row in the mapping reference record having a Target Panel value of BTN0ENSY, which is the third row listed in the Mapping Reference panel on page 166. The Target Field/Command and Source Prefix fields in this row are empty. On finding these fields empty, the PostProcessor looks for the Tivoli Information Management for z/OS message BLG03058I (Record ________ stored successfully).

11. If the PostProcessor finds message BLG03058I, it issues a message that indicates successful post-processing. The AutoBridge-created record is deleted and the PostProcessor moves on to the next record in the search results list.

12. After all records have been post-processed, the PostProcessor waits for AutoBridge to create a new record.

## Installing the PostProcessor

Many of the steps for installing the PostProcessor are similar to those for installing Tivoli Information Management for z/OS. Where further information is needed about Tivoli Information Management for z/OS and its installation and maintenance utilities, refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*.

Remember to verify each step before proceeding to the next. Table 30 is a checklist for you to use as you complete each step of the planning process.

*Table 30. PostProcessor planning checklist*

| ✓ | Step |
|---|---|
| | “Step 1. Plan for the PostProcessor Panels” on page 168 |
| | “Step 2. Update Your Tivoli Information Management for z/OS Session Member” on page 169 |
| | “Step 3. Create a TSO Background Procedure for the PostProcessor” on page 169 |
| | “Step 4. Create Background Profiles for the PostProcessor” on page 170 |

## Step 1. Plan for the PostProcessor Panels

Installing the PostProcessor requires that you examine five panels. If you are running a previous version of the PostProcessor, it is not necessary to make any changes to your existing panels. If you have customized any of the following panels, you should compare these panels to the versions of the panels as they were shipped from Tivoli and restore onto your customized panels any aspects that you may have modified from the Tivoli-supplied versions:

BLG00011
BLG00010
BLG1ACOP
BLG1A210

If you have not customized these panels, you can use the versions supplied with the Tivoli Information Management for z/OS PostProcessor and skip this section.

In addition to those four panels, if you modified panel BLGAPI00, you must modify that panel in order to use the PostProcessor. BLGAPI00 is the controlling panel for the Tivoli Information Management for z/OS API, and, as shipped, has the PostProcessor exit EYMSP010 disabled. You can enable it by deleting the BRANCH control line that precedes the USEREXIT line for EYMSP010. See “Modifying Panel BLGAPI00” on page 171 and review the comment lines.

**Note:** An experienced Tivoli Information Management for z/OS operator should perform the necessary modifications.

The PostProcessor panels are included in the Tivoli Information Management for z/OS base panel data set, and you do not need to load them separately. This is a list of the PostProcessor panels included in SBLMPNLS:

```
EYMAM100   EYMBM100   EYMBM500   EYM1MBAK   EYM1MCOP   EYM1M500
EYM1M501   EYM1M92A   EYM1M92B   EYM1M921   EYM1M925   EYM1M926
EYM1M928   EYM1M929   EYM2M100   EYM2M110   EYM2M120   EYM2M130
EYM2M200   EYM2M250   EYM2M500   EYM2M510   EYM4M901   EYM4M902
EYM4M903   EYM4M904   EYM5MARK   EYM5M100   EYM5M110   EYM5M200
EYM5M250   EYM5M500   EYM6ACCN   EYM6ALTD   EYM6ALTT   EYM6CLAE
EYM6CRDT   EYM6CRTM   EYM6DSAB   EYM6MARK   EYM6POWN   EYM6RIDN
EYM6RNOD   EYM6SPFX   EYM6TFLD   EYM6TPAN   EYM6TPNI   EYM6URN0
EYM6USER   EYM6USRE   EYM9MAIN   EYM9MAPB   EYM9MAPE   EYM9M500
EYM9M7XC   EYM9M7XI   EYM9M901   EYM9M902   EYM9M903   EYM9M904
EYM9M904   EYM9POST   EYM9RSET   EYM9XMIT
```

## Step 2. Update Your Tivoli Information Management for z/OS Session Member

If you do not want all of your API sessions to run the PostProcessor, you should create a VSAM panel data set that contains a copy of BLGAPI00 that has been modified to invoke the PostProcessor. This panel data set should be placed into the RPANELS concatenation of a session member before any other copy of BLGAPI00. This session member should be accessed only by the NetView Bridge Adapter proc (or other API application) that you wish to have records post-processed.

## Step 3. Create a TSO Background Procedure for the PostProcessor

An instance of the PostProcessor starts as a background procedure the first time AutoBridge passes a Tivoli Information Management for z/OS record through the Tivoli Information Management for z/OS NetView Bridge Adapter, or the first time a record passes through the Tivoli Information Management for z/OS API using a session that calls the PostProcessor. This is an example procedure for starting the PostProcessor. The data set names in the figure are examples; replace them with names appropriate to your installation.

```
//**********************************************************************
//*                                                                    *
//* NAME: EYMPOST (PostProcessor PROC)                                 *
//*                                                                    *
//* PURPOSE: START THE POSTPROCESSOR                                   *
//*                                                                    *
//* Change blm.v0r0m0 to your Information Management for z/OS qualifier *
//* Change eyl.bridge to your PostProcessor qualifier.                 *
//* Change eyl.&ppid  to your PostProcessor ID (PP00 - PP15)      *
//*                                                                    *
//* Change the other qualifiers to match your ISPF installation.       *
//*                                                                    *
//**********************************************************************
//EYMPROC PROC
//*
//EYMPOST EXEC PGM=EYMSP020,REGION=4096K
//STEPLIB  DD DSN=blm.v0r0m0.SBLMMOD1,DISP=SHR
//*
//BLGTSX   DD DSN=blm.v0r0m0.SBLMTSX,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD DSN=eyl.bridge.SYSTSPRT,DISP=SHR
//SYSTSIN  DD DSN=eyl.bridge.SYSTSIN,DISP=SHR
//ISPPROF  DD DSN=eyl.&ppid..ISPPROF,DISP=SHR
//ISPPLIB  DD DSN=blm.v0r0m0.SBLMSAMP,DISP=SHR
//         DD DSN=ISR.ISRPLIB,DISP=SHR
//         DD DSN=ISP.ISPPLIB,DISP=SHR
//ISPTLIB  DD DSN=ISR.ISRTLIB,DISP=SHR
//         DD DSN=ISP.ISPTLIB,DISP=SHR
//ISPMLIB  DD DSN=ISR.ISRMLIB,DISP=SHR
//         DD DSN=ISP.ISPMLIB,DISP=SHR
//ISPSLIB  DD DSN=ISR.ISRSLIB,DISP=SHR
//         DD DSN=ISP.ISPSLIB,DISP=SHR
//ISPLLIB  DD DSN=blm.v0r0m0.SBLMMOD1,DISP=SHR
//         DD DSN=ISR.ISRLOAD,DISP=SHR
//         DD DSN=ISP.ISPLOAD,DISP=SHR
//SYSPROC  DD DSN=ISR.ISRCLIB,DISP=SHR
```

This procedure is shipped in the EYMPOST member of the SBLMSAMP library. You must place EYMPOST in a data set that resides in an accessible PROCLIB. Do not rename it. Consult with your systems programmer for the correct placement of this member.

The following data definitions are likely to be allocated differently from the way they are currently allocated in your installation:

**STEPLIB** The STEPLIB DD statement must reference the data set containing the SBLMMOD1 loadlib unless it is in the link list concatenation.

**Note:** The PostProcessor code library is not re-entrant and is in loadlib SBLMMOD1. If you want to place SBLMMOD1 in the system link pack area (LPA), all jobs (NetView AutoBridge and background postprocessors) must have SBLMMOD1 in their STEPLIB. Otherwise an ABEND0C4 will occur when an attempt is made to use a PostProcessor load module (EYMSPnnn).

**SYSTSPRT** The SYSTSPRT DD statement identifies the destination of output messages issued by TSO.

**SYSTSIN** The SYSTSIN DD statement must reference the data set containing the TSO commands necessary to invoke Tivoli Information Management for z/OS and start the PostProcessor TSP. These commands must appear as follows:

```
PROFILE PREFIX(nnnnnn)
ISPSTART PGM(BLGINIT) PARM(SESS(00) IRC(RUN EYM9MAIN))
```

The following considerations apply to these commands:

- You must set the PREFIX parameter (the *nnnnnn* in the example above) of the PROFILE command to an authorized high-level qualifier on your system. Tivoli Information Management for z/OS may use this value to allocate temporary data sets during PostProcessor execution.
- The SESS parameter of the ISPSTART command must refer to a session member that accesses the same database and panels that AutoBridge and your Tivoli Information Management for z/OS operators access.
- You can also use the CLASS parameter instead of specifying an invocation class in the background profile. See “Step 4. Create Background Profiles for the PostProcessor” for information on specifying a PostProcessor privilege class.
- Using the other Tivoli Information Management for z/OS ISPSTART parameters is not recommended.

**ISPPROF** The ISPPROF DD statement must reference a data set name containing the *&ppid.* insert as one of its middle-level qualifiers. When starting this procedure, the PostProcessor passes a value in this parameter, PP*nn*, that is derived from a count of how many PostProcessor tasks are active on the system (*nn* = 00 through 15). See “Step 4. Create Background Profiles for the PostProcessor” for information on defining ISPF profiles for the PostProcessor.

**ISPLLIB** The ISPLLIB DD statement should also reference the data set containing the SBLMMOD1 code library.

## Step 4. Create Background Profiles for the PostProcessor

While processing records, the PostProcessor uses an ISPF profile just as an operator would. This profile is in member BLG0PROF of the partitioned data set allocated to the ISPPROF DD statement in the background procedure that starts the PostProcessor task. You must create a unique profile data set for up to 16 PostProcessor tasks running at the same time. The maximum number of PostProcessor tasks running must be less than or equal to the number of simultaneously running NetView Bridge adapters. If you have other Tivoli

Information Management for z/OS API applications that are exploiting the PostProcessor, they must be considered when determining the maximum number of PostProcessor tasks that can run at the same time.

Each address space containing an initialized Tivoli Information Management for z/OS API environment will cause a PostProcessor task to be started at the first call to TSP user exit EYMSP010. As profile data sets do not occupy significant space, you should allocate the maximum number (16) while performing these installation tasks. When allocating these data sets, consider the placement of PP00 through PP15 in the EYMPOST procedure. If your installation data set naming conventions permit you to follow these installation instructions explicitly, then allocate 16 data sets:  EYL.PP00.ISPPROF through EYL.PP15.ISPPROF.

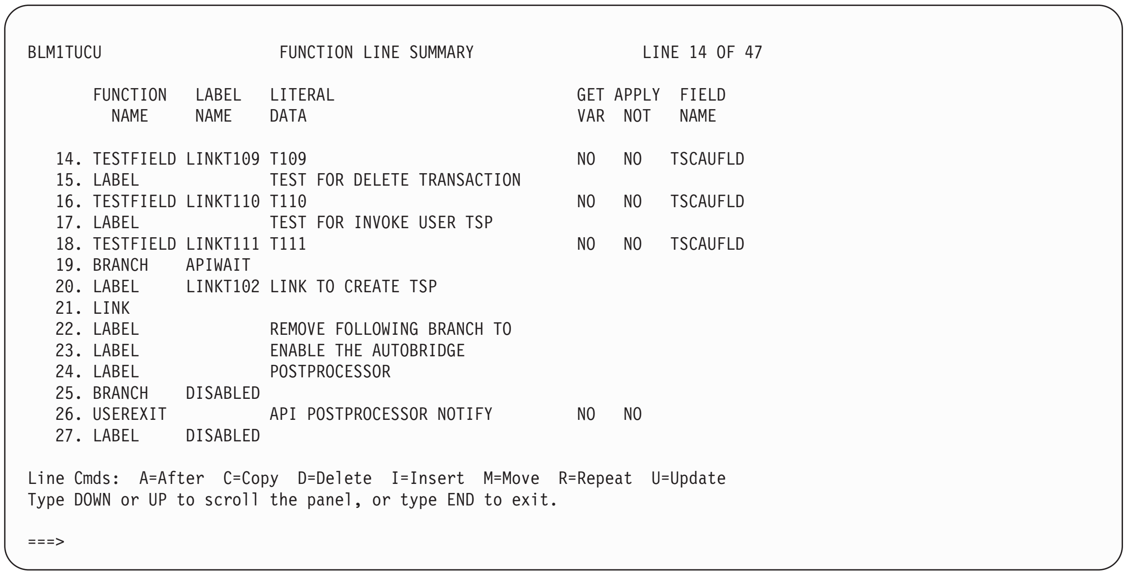
You can load each profile by copying member BLG0PROF from your profile data set, or you can create a custom profile for the PostProcessor as follows:

1. In your profile data set, copy member BLG0PROF to a new member name.
2. Invoke Tivoli Information Management for z/OS and define the profile settings for the PostProcessor using the Tivoli Information Management for z/OS PROFILE command. Command detection must be set to DATA and the invocation class must be set to a privilege class name with database administration authority and full access authority to the records you will be post-processing. Do not specify an invocation SRC.
3. Exit from Tivoli Information Management for z/OS using the QUIT command.
4. Copy member BLG0PROF from your profile data set into the profile data sets for the PostProcessor.
5. Delete member BLG0PROF from your profile data set.
6. Rename the member you created in step 1 above to BLG0PROF.

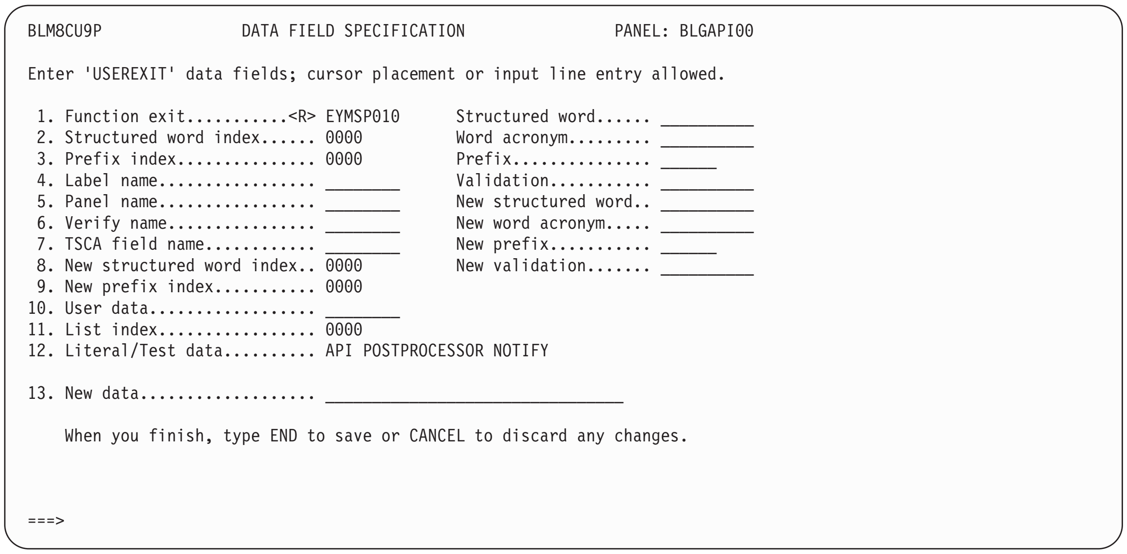
Your PostProcessor profiles are now ready.

## Modifying Panel BLGAPI00

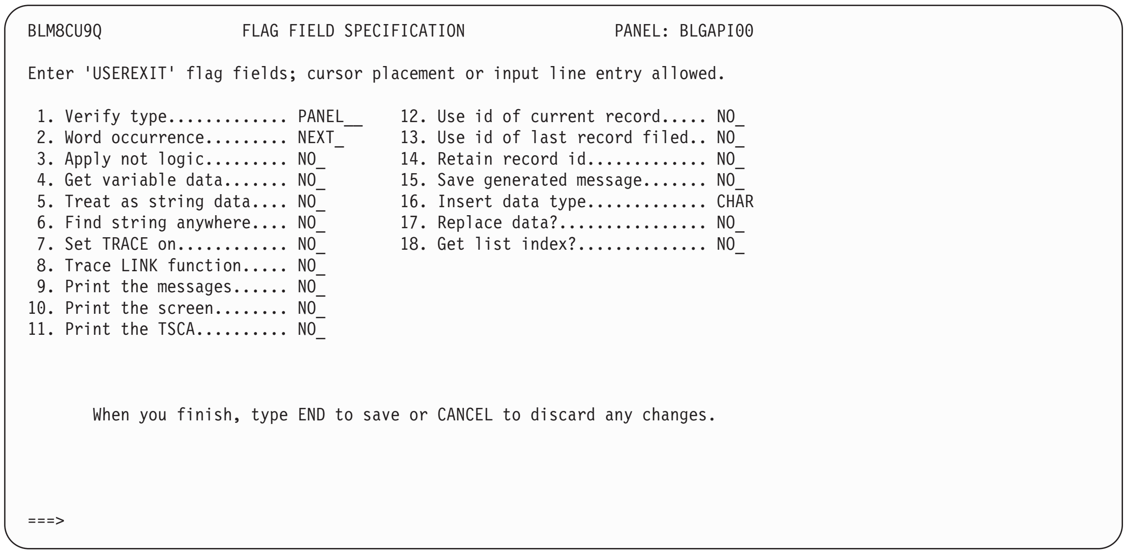
Modify panel BLGAPI00 to delete the BRANCH control line that calls exit EYMSP010 to invoke the PostProcessor (see the following example), or if you have modified BLGAPI00, make the changes documented here. Place this control line in the TSP so that it runs only after the Tivoli Information Management for z/OS API CREATE function has completed.



```
BLM1TUCU                  FUNCTION LINE SUMMARY                 LINE 14 OF 47

       FUNCTION   LABEL   LITERAL                          GET APPLY  FIELD
         NAME     NAME    DATA                             VAR  NOT   NAME

   14. TESTFIELD LINKT109 T109                             NO   NO   TSCAUFLD
   15. LABEL              TEST FOR DELETE TRANSACTION
   16. TESTFIELD LINKT110 T110                             NO   NO   TSCAUFLD
   17. LABEL              TEST FOR INVOKE USER TSP
   18. TESTFIELD LINKT111 T111                             NO   NO   TSCAUFLD
   19. BRANCH    APIWAIT
   20. LABEL     LINKT102 LINK TO CREATE TSP
   21. LINK
   22. LABEL              REMOVE FOLLOWING BRANCH TO
   23. LABEL              ENABLE THE AUTOBRIDGE
   24. LABEL              POSTPROCESSOR
   25. BRANCH    DISABLED
   26. USEREXIT           API POSTPROCESSOR NOTIFY         NO   NO
   27. LABEL     DISABLED

Line Cmds:  A=After  C=Copy  D=Delete  I=Insert  M=Move  R=Repeat  U=Update
Type DOWN or UP to scroll the panel, or type END to exit.

===>
```



```
BLM8CU9P              DATA FIELD SPECIFICATION              PANEL: BLGAPI00

Enter 'USEREXIT' data fields; cursor placement or input line entry allowed.

 1. Function exit..........<R> EYMSP010      Structured word...... __________
 2. Structured word index...... 0000         Word acronym......... __________
 3. Prefix index.............. 0000          Prefix.............. ______
 4. Label name................ ________       Validation.......... __________
 5. Panel name................ ________      New structured word.. __________
 6. Verify name............... ________      New word acronym..... __________
 7. TSCA field name........... ________      New prefix........... ______
 8. New structured word index.. 0000         New validation....... __________
 9. New prefix index.......... 0000
10. User data................. ________
11. List index................ 0000
12. Literal/Test data......... API POSTPROCESSOR NOTIFY

13. New data.................. ________________________________

    When you finish, type END to save or CANCEL to discard any changes.



===>
```



```
BLM8CU9Q              FLAG FIELD SPECIFICATION              PANEL: BLGAPI00

Enter 'USEREXIT' flag fields; cursor placement or input line entry allowed.

 1. Verify type............ PANEL__    12. Use id of current record..... NO_
 2. Word occurrence......... NEXT_     13. Use id of last record filed.. NO_
 3. Apply not logic......... NO_       14. Retain record id............ NO_
 4. Get variable data....... NO_       15. Save generated message....... NO_
 5. Treat as string data.... NO_       16. Insert data type............ CHAR
 6. Find string anywhere.... NO_       17. Replace data?............... NO_
 7. Set TRACE on........... NO_        18. Get list index?............. NO_
 8. Trace LINK function..... NO_
 9. Print the messages...... NO_
10. Print the screen........ NO_
11. Print the TSCA.......... NO_



       When you finish, type END to save or CANCEL to discard any changes.




===>
```

## Setting Up the PostProcessor

This section describes the planning and setup tasks necessary to start the PostProcessor. It also helps you determine what information needs to be post-processed and contains instructions for building and maintaining mapping reference records.

### Determining If a Record Should Be Post-Processed

The following records are appropriate candidates for post-processing:

- Records entered on any panels containing any fields or selections that trigger program exits or TSPs to supplement or modify data either within the record or in some other record, database, or application.
- Records that, when filed, invoke the Tivoli Information Management for z/OS notification facility to send a message to one or more operators.

Records created by AutoBridge will not be subject to these processes unless they are accessed and handled by the PostProcessor.

### Determining Which Fields or Selections to Post-Process

Your Tivoli Information Management for z/OS administrator must be familiar with the internals of your customized Tivoli Information Management for z/OS panels to identify the selections and fields that require post-processing. If such information is not documented or readily available, you can use the Tivoli Information Management for z/OS FLOW command to identify any control panels or TSPs that may be called to supplement or modify record data. Only an experienced Tivoli Information Management for z/OS operator should perform this task.

### Authorizing the PostProcessor to Tivoli Information Management for z/OS

During processing, the PostProcessor reads mapping reference records and also copies, updates, and deletes records created by AutoBridge. Therefore, you must be sure that the PostProcessor is authorized to perform these types of record operations. Because the PostProcessor runs as an MVS-started task, it assumes an assigned task ID as its TSO user ID. The TSO ID could be the default value (STC) or could be defined based on the setup of your installation. If you are a user of the IBM Resource Access Control Facility (RACF), define this ID in the ICHRIN03 member of SYS1.LINKLIB. Consult your MVS systems programmer for the TSO user ID that will be assigned to the PostProcessor task.

To determine the task user ID, allow the PostProcessor to start with at least one AutoBridge-created record in the database. Without proper Tivoli Information Management for z/OS authority, the PostProcessor fails and Tivoli Information Management for z/OS error messages are written to the destination specified by the SYSPRINT DD statement. When this occurs, look for the message

```
BLG10030I (Logon identifier ______ is not in privilege class _______)
```

The message will contain the user ID assigned to the PostProcessor task.

You must enter the ID for the PostProcessor task as an eligible user in the privilege class in which the PostProcessor operates. This privilege class must have database administration authority to access mapping reference records in addition to whatever authority is necessary to post-process the AutoBridge-created records.

## Modifying the Tivoli Information Management for z/OS Profile

When accessing mapping reference records, your Tivoli Information Management for z/OS profile must have command detection set to DATA; otherwise, unexpected errors may occur.

## Authorizing User Access to Mapping Reference Records

Mapping reference records contain sensitive data that is critical to the integrity of any records processed by the PostProcessor. Mapping reference records should be created or modified only by those who fully understand record processing in the Tivoli Information Management for z/OS database. Such personnel should know the commands and selections necessary to permit the PostProcessor to create a complete Tivoli Information Management for z/OS record from an AutoBridge-created record.

Mapping reference record dialogs are designed to allow only those Tivoli Information Management for z/OS operators with database administration authority to create, access, or manipulate mapping reference records.

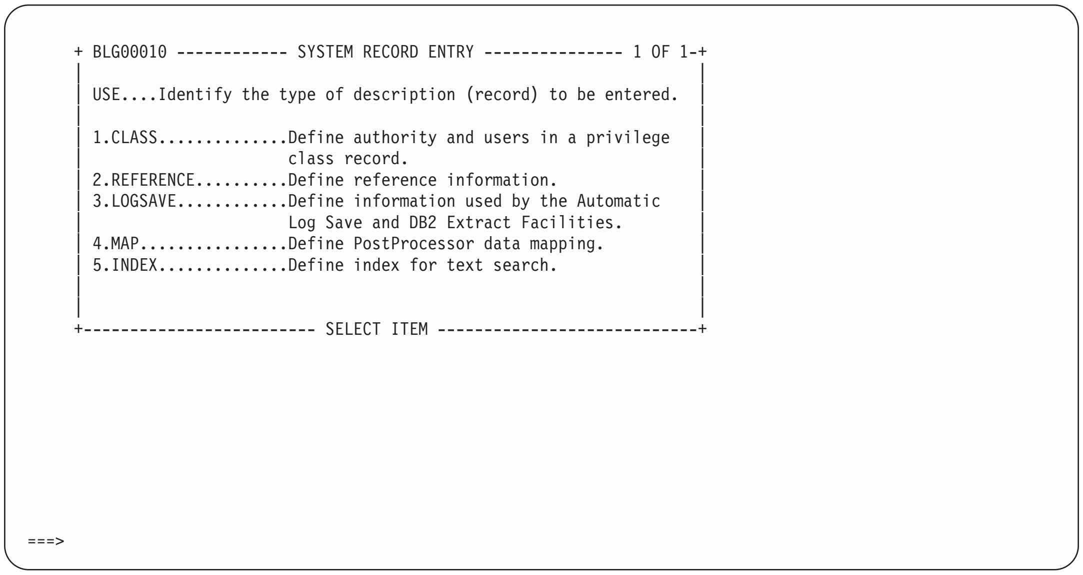
To authorize an operator to access mapping reference records, do the following:

1. Define a privilege class record that has database administration authority enabled (set to YES).
2. Add the TSO user IDs of those authorized to access mapping reference records to the list of eligible users in the privilege class record.
3. Users working with mapping reference records must make this authorized privilege class the current class for their session.

## Creating a Mapping Reference Record

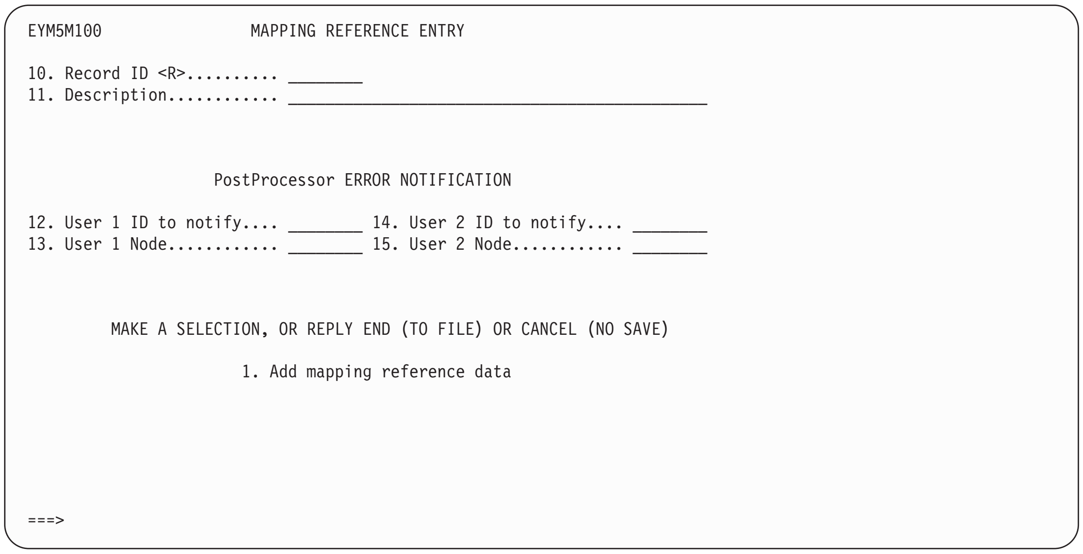
The following procedure describes the steps involved in creating a mapping reference record:

1. From the System Primary Options Menu (BLG0EN10), select option 5 (ENTRY). The System Record Entry panel (BLG00010) is displayed.



```
+ BLG00010 ------------ SYSTEM RECORD ENTRY --------------- 1 OF 1-+
|                                                                   |
| USE....Identify the type of description (record) to be entered.  |
|                                                                   |
| 1.CLASS.............Define authority and users in a privilege     |
|                     class record.                                 |
| 2.REFERENCE.........Define reference information.                 |
| 3.LOGSAVE...........Define information used by the Automatic      |
|                     Log Save and DB2 Extract Facilities.          |
| 4.MAP...............Define PostProcessor data mapping.            |
| 5.INDEX.............Define index for text search.                 |
|                                                                   |
|                                                                   |
+------------------------- SELECT ITEM ----------------------------+

===>
```

2. From the Reference Entry selection panel (BLG00010), select option 4 (MAP). The Mapping Reference Entry panel (EYM5M100) is displayed as shown in this example.



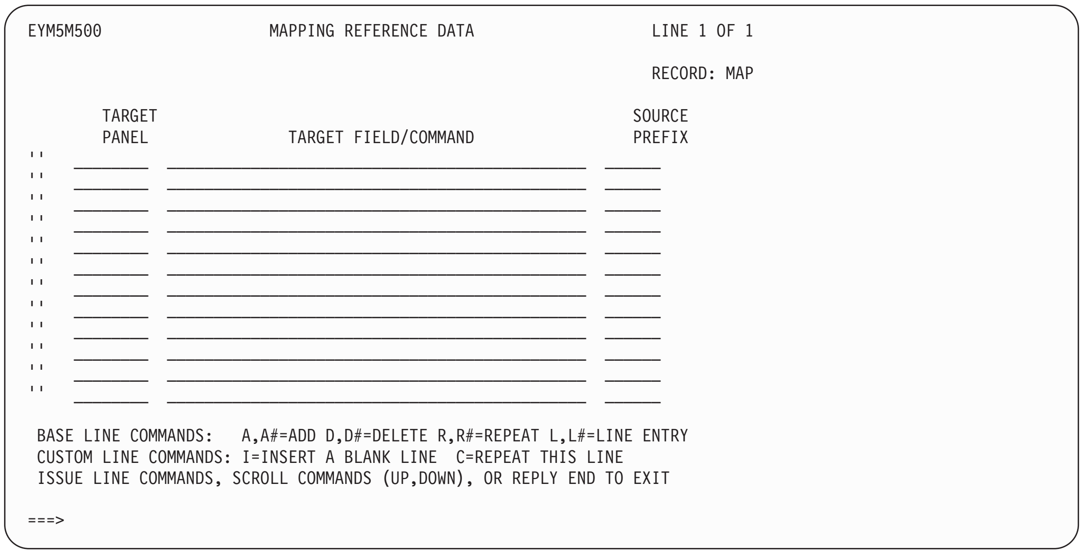
```
EYM5M100               MAPPING REFERENCE ENTRY

10. Record ID <R>.......... ________
11. Description............ _____________________________________________



                    PostProcessor ERROR NOTIFICATION

12. User 1 ID to notify.... ________ 14. User 2 ID to notify.... ________
13. User 1 Node............ ________ 15. User 2 Node............ ________



         MAKE A SELECTION, OR REPLY END (TO FILE) OR CANCEL (NO SAVE)

                        1. Add mapping reference data






===>
```

This is the main entry panel for the mapping reference record. You are required to enter the record ID of a mapping reference record in field 10. The record ID you enter must be unique in the database and match the ID that is entered in the record by AutoBridge.

Fields 12 through 15 allow you to enter node and user ID information to notify two operators about PostProcessor failures.

To get more information about the fields on this panel, enter **HELP** on the command line.

3. Select option 1 (Add mapping reference data) to specify the fields, commands, and selections to be processed by the PostProcessor. The Mapping Reference Data entry panel (EYM5M500) is displayed as shown in this example.



```
EYM5M500                 MAPPING REFERENCE DATA                 LINE 1 OF 1

                                                                RECORD: MAP

        TARGET                                                SOURCE
        PANEL              TARGET FIELD/COMMAND               PREFIX
''    ________  _____________________________________________  ______
''    ________  _____________________________________________  ______
''    ________  _____________________________________________  ______
''    ________  _____________________________________________  ______
''    ________  _____________________________________________  ______
''    ________  _____________________________________________  ______
''    ________  _____________________________________________  ______
''    ________  _____________________________________________  ______
''    ________  _____________________________________________  ______
''    ________  _____________________________________________  ______
''    ________  _____________________________________________  ______
''    ________  _____________________________________________  ______

 BASE LINE COMMANDS:   A,A#=ADD D,D#=DELETE R,R#=REPEAT L,L#=LINE ENTRY
 CUSTOM LINE COMMANDS: I=INSERT A BLANK LINE  C=REPEAT THIS LINE
 ISSUE LINE COMMANDS, SCROLL COMMANDS (UP,DOWN), OR REPLY END TO EXIT

===>
```

The information you enter on this panel determines how the PostProcessor processes your AutoBridge-created records. Information on entering mapping reference record data follows this procedure. To see detailed information about each field on this panel, enter **HELP** on the command line. The line commands on this panel work the same as those on standard Tivoli Information Management for z/OS panels.

4. When you have finished entering mapping reference record data on this panel, exit the panel as follows:
   - To save the changes to this mapping reference record, enter **END** on the command line.
   - To quit without saving your changes, enter **CANCEL**.

   Either action returns you to the Mapping Reference Entry panel (EYM5M100).
5. Close the entry session for this mapping reference record as follows:
   - To file the record in the database, enter **END** on the command line.
   - To cancel entry of the record, enter **CANCEL**.

   Either action returns you to the Information/System Primary Options Menu (BLG0EN10).

When defining the PostProcessor commands and selections for a mapping reference record, think of the PostProcessor as an automated end user. Combine the selections and fields used for post-processing with any selections and commands necessary for navigating through your Tivoli Information Management for z/OS panels and filing the processed record. "PostProcessor Example" on page 164 shows the mapping reference record contents.

## Mapping Reference Record Considerations

Keep in mind the following considerations when creating mapping reference records:

- The PostProcessor determines which mapping reference record to use by reading the value in the transaction record that corresponds to the PostProcessor structured word specified in the alias table. This value is the record ID of the mapping reference record to be used. If there is no such value specified in the transaction record, the default value specified in the alias table is used.
- The PostProcessor copies the AutoBridge-created record using the Tivoli Information Management for z/OS COPY command; therefore, the starting panel is the same as if you had initiated a record copy from your terminal.
- The PostProcessor keeps track of the rows it has used. Each row in the mapping reference record that is accessed by the PostProcessor is used only once for each record processed.
- Rows that contain the names of Target Panels that are not accessed during record processing are ignored. Such entries do not cause post-processing errors.
- Mapping reference records must include a selection or command to file the post-processed record.
- The PostProcessor determines its success by testing for the following message:
  ```
  BLG03058I Record ________ stored successfully
  ```
  The PostProcessor looks for this message when it locates a row in the mapping reference record having a value in the Target Panel column and no values in the Target Field/Command and Source Prefix columns. The Target Panel value in this row will typically be the name of a primary options menu such as Tivoli Information Management for z/OS's BLG0EN20. The value you use may differ based on the customization at your site.

- The PostProcessor can perform only limited processing of freeform text and does so only with the Tivoli Information Management for z/OS editor. Do not use the PostProcessor to manipulate freeform text.
- The PostProcessor encounters an error if it attempts to leave a panel that does not have all required fields filled. Either AutoBridge must create records with all required fields filled, or the mapping reference record data used during post-processing must contain commands that supply values to these fields.

**Note:** Verify your mapping reference record content manually as follows:
1. Print your mapping reference record.
2. Create a test AutoBridge record.
3. Manually follow the mapping reference record procedure in the same way as the PostProcessor would. Remember to begin by copying the record.

## Maintaining Mapping Reference Records

As the administrative processes at your site evolve, your Tivoli Information Management for z/OS applications will change. These changes may affect panel flows and the hidden automation behind fields and selections. As a result, you may need to modify your mapping reference records periodically.

### Modifying Mapping Reference Records

You can access and modify mapping reference records like any other Tivoli Information Management for z/OS record. All record operations are supported for mapping reference records (update, display, copy, purge, print) and can be initiated using the typical methods:
- Enter a command on the command line.
- Use the UTILITY dialog (Option 7 on the primary options menu).
- Enter line commands next to mapping reference record entries in a search results list.

Helpful details about record operations follow.

- When you copy a mapping reference record, all field values are carried into the new record with the exception of the following:
  Record ID
  Date entered
  Time entered
  Entry privilege class
  User ID last altered
  Date last altered
  Time last altered
  Owning privilege class
- When you print a mapping reference record, output is generated that includes the complete record contents, formatted according to RFT EYMPRMR.

### Locating Mapping Reference Records

Search for mapping reference records using the standard Tivoli Information Management for z/OS methods:

- Enter freeform SEARCH commands directly on the command line or via the ARGUMENT command.
- Create structured searches using the inquiry dialog (Option 6) from the System Primary Options Menu (BLG0EN10).

- Use a combination of these methods.

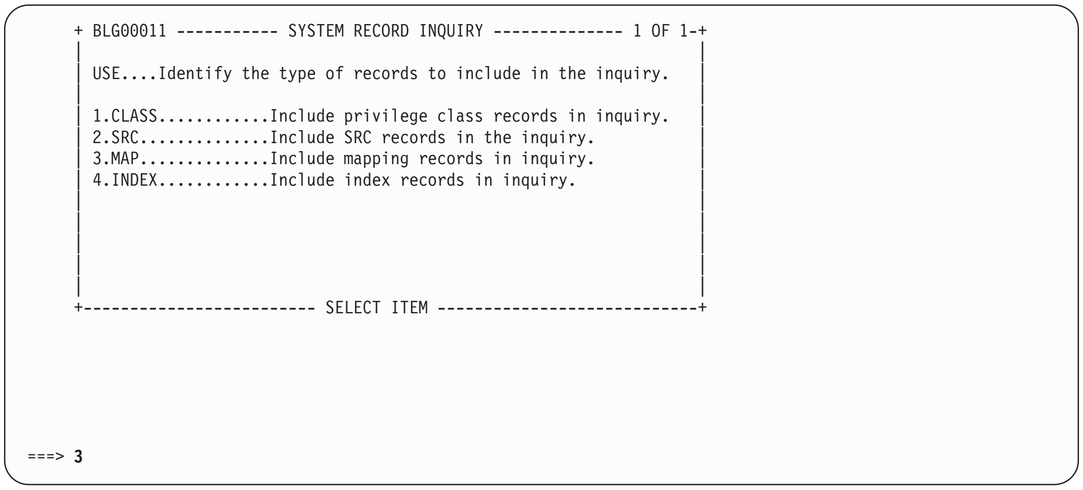
As shipped, the following fields will be searchable (recognized) in the mapping reference record:

Record ID
User 1 ID to notify
User 1 Node
User 2 ID to notify
User 2 Node
Date entered
Time entered
Entry privilege class
User ID last altered
Date last altered
Time last altered
Owning privilege class
Target panel (column in mapping reference list)

In the System dialog, only the Quick Search Panels path is supported for structured searches. Therefore, set the Quick Search Panels value to YES in your profile. Otherwise you may get inaccurate search results.

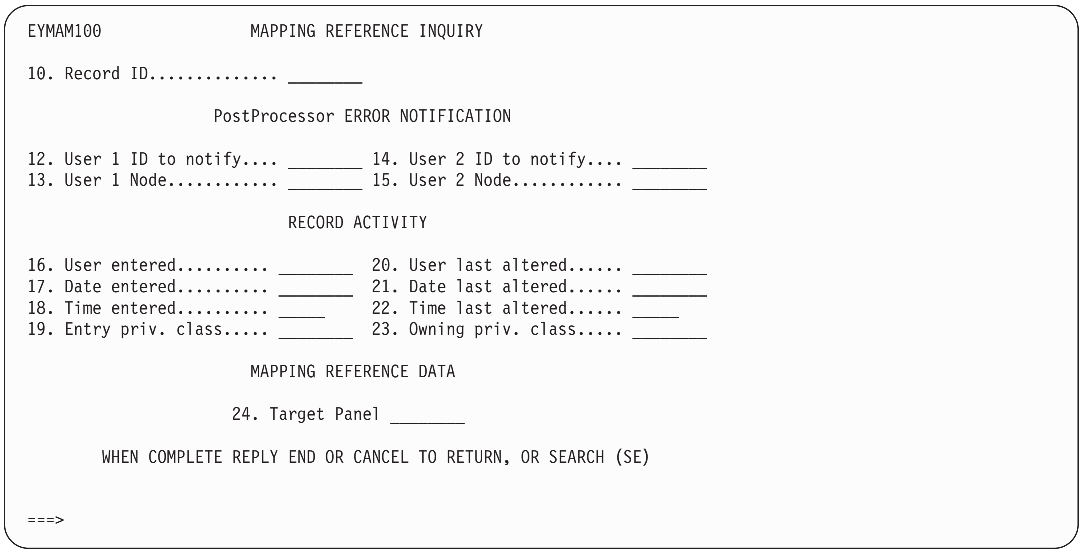
To locate a mapping reference record, perform a structured search as follows:

1. From the System Primary Options Menu (BLG0EN10), select option 6 (INQUIRY). The System Inquiry selection panel (BLG00011) is displayed as shown in this example. Type **3** for mapping records and press Enter.



```
      + BLG00011 ----------- SYSTEM RECORD INQUIRY -------------- 1 OF 1-+
      |                                                                  |
      | USE....Identify the type of records to include in the inquiry.   |
      |                                                                  |
      | 1.CLASS............Include privilege class records in inquiry.   |
      | 2.SRC..............Include SRC records in the inquiry.           |
      | 3.MAP..............Include mapping records in inquiry.           |
      | 4.INDEX............Include index records in inquiry.             |
      |                                                                  |
      |                                                                  |
      |                                                                  |
      |                                                                  |
      |                                                                  |
      +------------------------- SELECT ITEM ----------------------------+





 ===> 3
```

2. The mapping reference record Inquiry panel (EYMAM100) is displayed as shown here.



```
EYMAM100               MAPPING REFERENCE INQUIRY

10. Record ID............. _______

                     PostProcessor ERROR NOTIFICATION

12. User 1 ID to notify.... _______ 14. User 2 ID to notify.... _______
13. User 1 Node........... _______ 15. User 2 Node........... _______

                             RECORD ACTIVITY

16. User entered......... _______  20. User last altered...... _______
17. Date entered......... _______  21. Date last altered...... _______
18. Time entered......... ____     22. Time last altered...... ____
19. Entry priv. class..... _______  23. Owning priv. class..... _______

                         MAPPING REFERENCE DATA

                        24. Target Panel _______

        WHEN COMPLETE REPLY END OR CANCEL TO RETURN, OR SEARCH (SE)


===>
```

To view detailed information about each field on this panel, enter **HELP** on the command line.

Use this inquiry panel like you would use the other Tivoli Information Management for z/OS Quick Search panels. Once you have entered the search criteria, enter **SEARCH** to initiate the database search.

### Using Updated Mapping Reference Records

The PostProcessor uses the mapping reference record that is in memory until a different record is called. For the PostProcessor to use an updated mapping reference record, you must stop the PostProcessor task and then restart it. Do this by responding to message EYM000I with the END command. The next API event will restart the PostProcessor.

## Modifying AutoBridge's Tivoli Information Management for z/OS Interface

This section describes modifications that you must make to Tivoli Information Management for z/OS's PIDT and alias table. This section also describes a consideration regarding the modification of Tivoli Information Management for z/OS's record file TSP (BLGAPI00).

If you need additional information on the PIDT and alias table, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

### PIDT Modifications

You must modify the Program Interface Data Table (PIDT) used by AutoBridge to include a reference to the Tivoli Information Management for z/OS structured word that the PostProcessor uses to distinguish an AutoBridge-created record from other database records. Your PIDT must refer to panel EYM6MARK, supplied with the PostProcessor. This example highlights the addition to the PIDT segment for Problem Create.

```
TABLE NAME(BLGYPRC) USE(CREATE) CODE(0110) SEPARATOR(,);

  FIELD PANEL(BLG00000) INDEX(S0032)        /*  PROBLEM RECORD TYPE  */
        REQUIRED(Y) RCDSWORD(Y);
  /*                POSTPROCESSOR MARKER SWORD                       */
  FIELD PANEL(EYM6MARK) INDEX(S7C00);       /*  POSTPROC FLAG        */
  /* REPORTER DATA                                                   */
  FIELD PANEL(BLG0BU00) INDEX(S0CFC);       /*  REPORTER DATA        */
FIELD PANEL(BLG6REQN) INDEX(S0B59)      /*  REPORTER NAME        */
      REQUIRED(Y);
FIELD PANEL(BLG6RQDP) INDEX(S0B9B);     /*  REPORTER DEPARTMENT  */
FIELD PANEL(BLG6PHON) INDEX(S0B2D);     /*  REPORTER PHONE       */
FIELD PANEL(BLG6OCCD) INDEX(S0C3D);     /*  DATE OCCURRED        */
FIELD PANEL(BLG6OCCT) INDEX(S0C6A);     /*  TIME OCCURRED        */
FIELD PANEL(BLG6NETN) INDEX(S0CA3);     /*  NETWORK NAME         */
FIELD PANEL(BLG6SYSN) INDEX(S0CA5);     /*  SYSTEM NAME          */
FIELD PANEL(BLG60APN) INDEX(S0CA8);     /*  PROGRAM NAME         */
FIELD PANEL(BLG6DEVN) INDEX(S0CA9);     /*  DEVICE NAME          */
FIELD PANEL(BLG6KIAF) INDEX(S0CBF);     /*  KEY ITEM AFFECTED    */
FIELD PANEL(BLG6REQD) INDEX(S0C49);     /*  DATE FIX REQUIRED    */
```

## Alias Table Modifications

As with the PIDT, you must modify the Tivoli Information Management for z/OS alias table to include an alias for the PostProcessor structured word, as shown in the following example. See the sample EYLALIAS in SBLMSAMP.

```
TABLE NAME(EYLALIA) USE(ALIAS);
ALIAS NAME(POSTPROC_FLAG)                /*  POSTPROCESSOR FLAG   */
      FIELD(S7C00)                       /*  POSTPROC FLAG SWORD  */
      DEFAULT(EYMMAPR);                  /*  OPTIONAL DEFAULT     */
ALIAS NAME(REPORTER_NAME)                /*  REPORTER NAME        */
      FIELD(S0B59)                       /*  REPORTER NAME SWORD  */
      DEFAULT(AUTOBRIDGE);               /*  DEFAULT REPORTER     */
ALIAS NAME(REPORTER_DEPT)                /*  REPORTER DEPT        */
      FIELD(S0B9B);                      /*  REPORTER DEPT SWORD  */
ALIAS NAME(REPORTER_PHONE)               /*  REPORTER PHONE       */
      FIELD(S0B2D);                      /*  REPORTER PHONE SWORD */
ALIAS NAME(DATE_OCCURRED)                /*  DATE OCCURRED        */
      FIELD(S0C3D);                      /*  DATE OCCURRED SWORD  */
ALIAS NAME(TIME_OCCURRED)                /*  TIME OCCURRED        */
      FIELD(S0C6A);                      /*  TIME OCCURRED SWORD  */
ALIAS NAME(NETWORK_NAME)                 /*  NETWORK NAME         */
      FIELD(S0CA3);                      /*  NETWORK NAME SWORD   */
ALIAS NAME(SYSTEM_NAME)                  /*  SYSTEM NAME          */
      FIELD(S0CA5);                      /*  SYSTEM NAME SWORD    */
ALIAS NAME(PROGRAM_NAME)                 /*  PROGRAM NAME         */
```

This example also shows an optional default value that can be specified in the alias table. All Tivoli Information Management for z/OS API applications with DEFAULT_OPTIONS set to ALL use this alias table to create records that are located and processed by the PostProcessor. If you have many applications that share the same alias table, do not specify a default value unless you want all API-created records to be processed by the PostProcessor using the same mapping reference record.

**Note:** If you specify a default value, it must be equal to the ID of a mapping reference record in the Tivoli Information Management for z/OS database. If you do not, then the API must supply the ID of a mapping reference record for the PostProcessor to use.

## Record File TSP Modifications

The PostProcessor is invoked when AutoBridge creates a record through Tivoli Information Management for z/OS's API. This is the result of the modification to BLGAPI00 discussed in "Step 1. Plan for the PostProcessor Panels" on page 168. If you have other applications that create records via the Tivoli Information Management for z/OS API, and you do not want to have the PostProcessor invoked in these instances, you need to set up your session members and panel data set concatenations so that only the AutoBridge session accesses the version of BLGAPI00 with the PostProcessor modifications.

## Running the PostProcessor

With proper installation, setup, and authorization, the PostProcessor runs automatically when AutoBridge enters a record into the Tivoli Information Management for z/OS database. The following sections discuss PostProcessor messages, error recovery, and halt conditions.

### Viewing PostProcessor Messages

All messages generated by the PostProcessor begin with the prefix EYM. PostProcessor messages are written to the system console and can be viewed directly via the Spool Display and Search Facility (SDSF) or any similar facility. Tivoli Information Management for z/OS messages that are indirectly generated by the PostProcessor are not collected, except for messages that occur when an unexpected error is encountered. Should such an error occur, these messages, along with an image of the last Tivoli Information Management for z/OS panel processed, are sent to the destination specified by the SYSPRINT DD statement defined to the PostProcessor.

### Recovering from PostProcessor Errors

Table 31 lists the most common PostProcessor errors, their causes, and the response necessary to correct them.

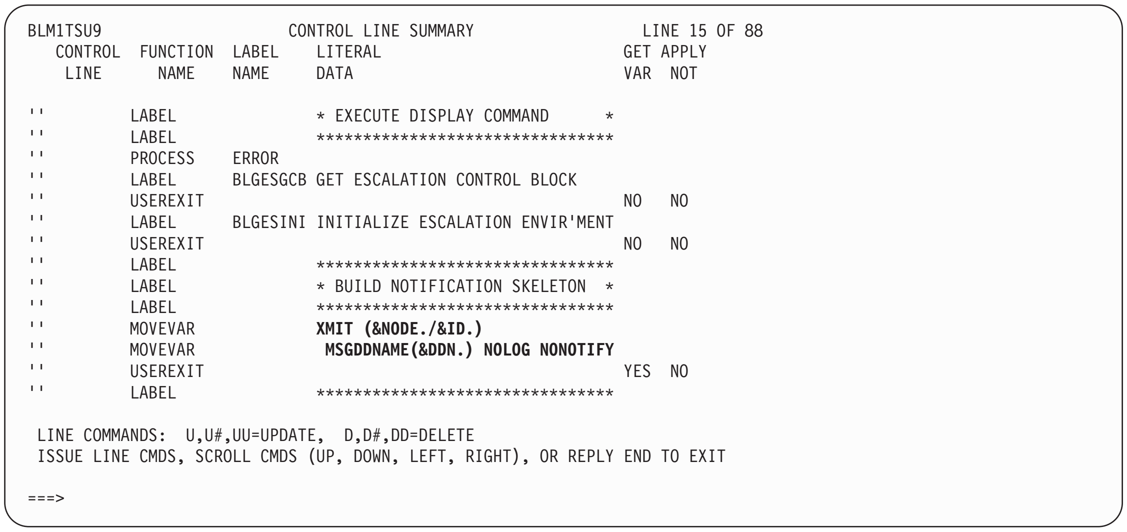
*Table 31. Probable causes and responses to common PostProcessor errors*

| Error | Probable cause | Corrective action |
|---|---|---|
| System '047' abends | One or more libraries are not authorized in the STEPLIB concatenation. | Verify that system authorization is complete for all data sets in the STEPLIB DD statements for the ADAPTER PROC and EYMPOST PROC. |
| Message BLG10030I | The PostProcessor may not have the correct authorization to access and modify records in the Tivoli Information Management for z/OS database. | Make sure that the PostProcessor is correctly authorized to read mapping reference records as well as copy, update, and purge records created by AutoBridge. See “Authorizing the PostProcessor to Tivoli Information Management for z/OS” on page 173 for instructions. |
| PostProcessor procedure EYMPOST does not start | Data sets not correctly defined. | Check data set allocations and concatenations to be sure the correct data sets are being referenced. In particular, check the ISPF profile data set, as its name is generated from a passed parameter and may not correlate to a data set name on the system. |
| Data validation errors<br>“Selection not found” messages<br>Panel flow errors<br>Other similar errors | Mapping reference record data is incorrectly specified. | Determine from the error messages which row in the mapping reference record data list is causing the error and make the necessary corrections. See “Creating a Mapping Reference Record” on page 174 for additional information on creating and validating a mapping reference record. |
| Message EYM107E | There is no mapping reference record in the database with an ID that is the same as that passed in by AutoBridge. | Create a mapping reference record with an ID that is the same as the ID passed by AutoBridge. If the Tivoli Information Management for z/OS alias table contains a default for the mapping reference record ID, make sure this value has been entered correctly. If there are customized AutoBridge command procedures that are overriding the default mapping reference record ID in the alias table, the procedures must have valid values so that the PostProcessor will not encounter an error. |

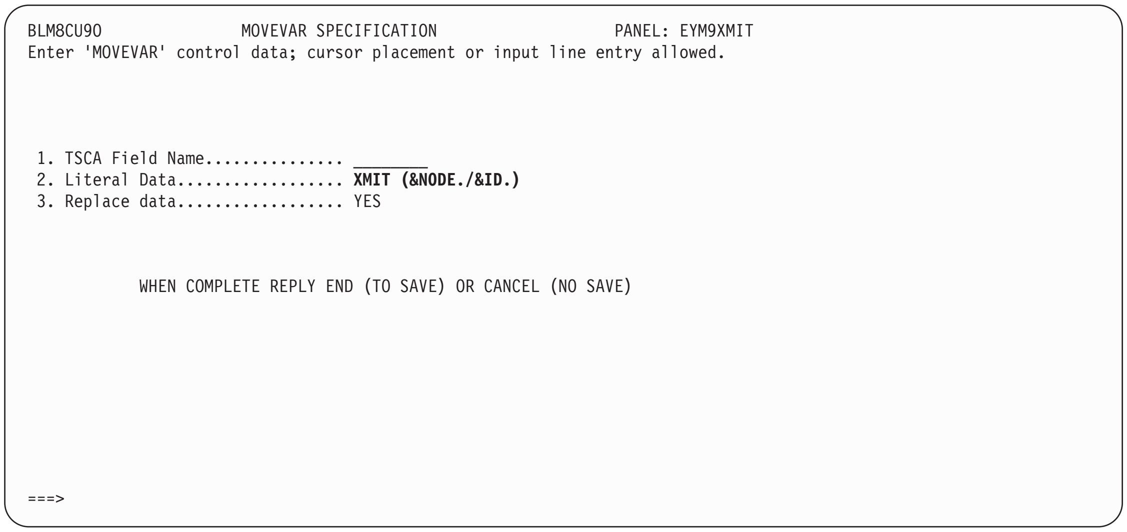
## Error Notification

When the PostProcessor encounters an error while attempting to process a record, it attempts to access the current mapping reference record and then to look for PostProcessor error-notification data. If it finds notification data, the PostProcessor will build an error message and transmit it to the user IDs specified in the mapping reference record using the TSO XMIT command.

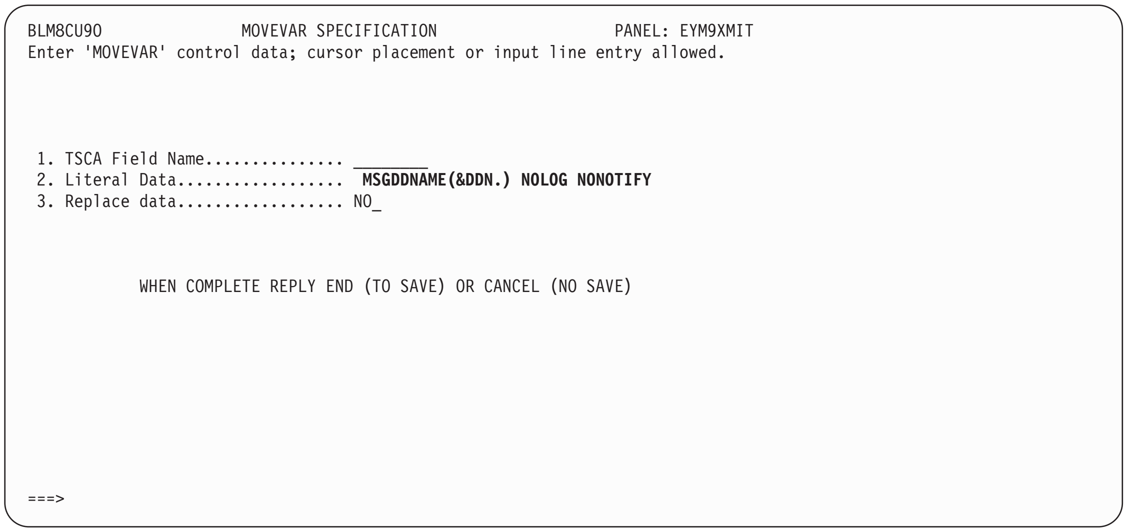
If your installation uses a different command or command format, you can change the command skeleton. The command skeleton is defined by a MOVEVAR control line in TSP EYM9XMIT and saved in the escalation control block (ESCB) as shown in this example. Refer to the *Tivoli Information Management for z/OS Problem, Change, and Configuration Management* or the *Tivoli Information Management for z/OS Panel Modification Facility Guide* for additional information.



```
BLM1TSU9                   CONTROL LINE SUMMARY                  LINE 15 OF 88
   CONTROL  FUNCTION  LABEL    LITERAL                          GET APPLY
    LINE      NAME    NAME     DATA                             VAR  NOT

''        LABEL              * EXECUTE DISPLAY COMMAND      *
''        LABEL              ********************************
''        PROCESS    ERROR
''        LABEL      BLGESGCB GET ESCALATION CONTROL BLOCK
''        USEREXIT                                          NO   NO
''        LABEL      BLGESINI INITIALIZE ESCALATION ENVIR'MENT
''        USEREXIT                                          NO   NO
''        LABEL              ********************************
''        LABEL              * BUILD NOTIFICATION SKELETON  *
''        LABEL              ********************************
''        MOVEVAR            XMIT (&NODE./&ID.)
''        MOVEVAR             MSGDDNAME(&DDN.) NOLOG NONOTIFY
''        USEREXIT                                          YES  NO
''        LABEL              ********************************

 LINE COMMANDS:  U,U#,UU=UPDATE,  D,D#,DD=DELETE
 ISSUE LINE CMDS, SCROLL CMDS (UP, DOWN, LEFT, RIGHT), OR REPLY END TO EXIT

===>
```



```
BLM8CU90              MOVEVAR SPECIFICATION                   PANEL: EYM9XMIT
Enter 'MOVEVAR' control data; cursor placement or input line entry allowed.




 1. TSCA Field Name............... ________
 2. Literal Data.................. XMIT (&NODE./&ID.)
 3. Replace data.................. YES



            WHEN COMPLETE REPLY END (TO SAVE) OR CANCEL (NO SAVE)








===>
```



```
BLM8CU90              MOVEVAR SPECIFICATION                 PANEL: EYM9XMIT
Enter 'MOVEVAR' control data; cursor placement or input line entry allowed.




 1. TSCA Field Name............... ________
 2. Literal Data.................. MSGDDNAME(&DDN.) NOLOG NONOTIFY
 3. Replace data.................. NO_



            WHEN COMPLETE REPLY END (TO SAVE) OR CANCEL (NO SAVE)










===>
```

## SNAP Macro Data

When system macros fail or internal logic errors occur, the PostProcessor calls the SNAP macro to create a dump of register and selected storage contents that you can use for problem determination. This data is sent to the destination specified by the SYSPRINT DD statement defined to the PostProcessor.

## Reprocessing Records in Error

Records that cause PostProcessor errors are flagged to ensure that another running PostProcessor does not locate these records in the database and attempt to reprocess them. After you identify and resolve such errors, you can reset the records so that the next PostProcessor that is activated will locate and process them.

Follow these steps to locate and update these records so that they will be reprocessed by the PostProcessor:

1. Enter Tivoli Information Management for z/OS using a privilege class with record update authority for the records to be processed.
2. Issue the command **RUN EYM9RSET** from the command line.

TSP EYM9RSET resets all records in the database that have encountered PostProcessor errors and makes the records available to the next PostProcessor that runs.

## Stopping the PostProcessor

The PostProcessor task stops running whenever one of the following occurs:

**A NetView Bridge adapter task ends**
The PostProcessor task associated with the adapter will end approximately one minute after the adapter address space is purged. This is the most likely cause of a PostProcessor task stopping. It does not require any operator intervention.

**You respond to console message EYM000I with the END command**
Do this only if you need to force a PostProcessor to stop prior to the purging of the NetView Bridge adapter or other application address space. If AutoBridge or an application associated with the address space continues to file more records, the PostProcessor will be automatically restarted.

**A severe PostProcessor error occurs**
Console messages and possibly a SNAP dump of memory contents will indicate the error condition. Also, the PostProcessor task is disabled so it does not attempt to restart.

## Mapping Reference Records Contents

Following is a list of the mapping reference record fields and their associated structured words and prefix words. Refer to the Assisted Entry panel for a complete list of the prefix words defined to each field.

*Table 32. Mapping reference record data model*

| Field name/Description | Assisted Entry Panel | S-word index | S-word | S-word index | S-word |
|---|---|---|---|---|---|
| Record type s-word | | 7C05 | XIM0I0MREC | | |
| Record ID | EYM6URN0 | | | 01E9 | RNID/ |
| Description | EYM6DSAB | 0E0F | XIM0TXCA00 | | |
| Target panel list item | EYM6TPAN | 7C10 | XIM0IL01 | 7B11 | TPAN/ |
| Target field list item | EYM6TFLD | 7C12 | XIM0IL02 | | |
| Source prefix list item | EYM6SPFX | 7C14 | XIM0IL03 | 7B04 | SPFX/ |
| Date entered | EYM6CRDT | 0C34 | XIM00SDC00 | 00D8 | DATE/ |
| Time entered | EYM6CRTM | 0C61 | XIM00STC00 | 028B | TIME/ |
| Entry privilege class | EYM6CLAE | 007A | XIM0I0CCS0 | 007A | CLAE/ |
| User ID entered | EYM6USRE | 7C0A | XIM0I0PE00 | 7B02 | USERE/ |
| Date last altered | EYM6ALTD | 0C35 | XIM00SDM00 | 00DF | DATM/ |
| Time last altered | EYM6ALTT | 0C62 | XIM00STM00 | 0291 | TIMM/ |
| Owning privilege class | EYM6POWN | 0BB5 | XIM0I0CCO0 | 007E | CLAO/ |
| User ID last altered | EYM6USER | 0B5E | XIM0I0PM00 | 02C4 | USER/ |
| User 1 ID to notify | EYM6RIDN | 0125 | XIM0I0PRL1 | 0468 | USRN/ |
| User 1 node | EYM6RNOD | 0126 | XIM0IC0RL1 | 00CA | COMX/ |
| User 2 ID to notify | EYM6RIDN | 0128 | XIM0I0PRL2 | 0468 | USRN/ |
| User 2 node | EYM6RNOD | 0129 | XIM0IC0RL2 | 00CA | COMX/ |

## AutoBridge PostProcessor User Exits

This section describes the AutoBridge PostProcessor user exits. For each user exit, the function, overview, description, input, output, and return codes are provided.

### EYMSP010

**MODULE NAME**
EYMSP010

**FUNCTION** API Event Detection Module Flow

**OVERVIEW** EYMSP010 API Event Detection Module
- Installed as an exit for BLGAPI00 TSP (file time TSP)
- Will GETMAIN storage for Common Program Storage
- Will start the PostProcessor job and pass CPS address

- After initial invocation will POST EYMSP030 module when there is processing to do

**DESCRIPTION**
EYMSP010 is activated when the API files a Tivoli Information Management for z/OS record. It will check for the PostProcessor being active, if it is, then POST EYMSP030's ECB and return to the TSP. If the PostProcessor is not active, then start EYMSP020, which is the PostProcessor initialization task. After the PostProcessor has been notified of work to be done then return control to the TSP.

**INPUT** None

**OUTPUT** None

**Return codes**
- 0 - no errors
- 4 - warning
- 8 - error
- 12 - severe error
- 16 - catastrophic error

## EYMSP020

**MODULE NAME**
EYMSP020

**FUNCTION** PostProcessor Main Task

**OVERVIEW** EYMSP020 PostProcessor Initialization task
- LOAD EYMSP030 program
- LINK to EYMSP030 task and pass address of CPS
- ATTACH TSO subtask (TSO will start an &im; session)
- WAIT on termination of TSO subtask
- LINK to EYMSP030 task to update status in CPS

**DESCRIPTION**
EYMSP020 will LOAD then LINK to EYMSP030 passing the address of the Common Program Storage area then ATTACH TSO as a sub-task. When control is returned after sub-task termination, perform address space clean-up and exit to MVS.

**INPUT** None

**OUTPUT** None

**Return codes**
- 0 - no errors
- 4 - warning
- 8 - error
- 12 - severe error
- 16 - catastrophic error

## EYMSP030

**MODULE NAME**
EYMSP030

**FUNCTION** EYMSP030 PostProcessor Link Module

**OVERVIEW** EYMSP030 PostProcessor Link Module
- Will BE LINKed to by EYMSP020 with CPS address
- POST EYMSP010 that PostProcessor is active
- Will BE LINKed to by EYMSP040
- Will issue a WTOR for termination
- WAIT to be POSTed by EYMSP010
- RETURN to EYMSP040 when there is work to do
- Pass termination command to EYMSP040
- Pass termination status to EYMSP010

**DESCRIPTION**

EYMSP030 is the link between the API address space and the PostProcessor batch address space. It is called by EYMSP020 for initilization/termination processing and by EYMSP040 to wait for work to do. EYMSP020 will pass the address of the CPS area which contains ECBs, ECB addresses and the ENQ name suffix which is used to determine the status of EYMSP010 and the PostProcessor. EYMSP030 will POST EYMSP010's ECB to indicate the PostProcessor is active and issue a WTOR message which will allow the operator to terminate the PostProcessor task. EYMSP040 will call EYMSP030 to wait for work and pass the status of the Tivoli Information Management for z/OS segment of the PostProcessor. EYMSP030 will WAIT to be POSTed by EYMSP010 when an API entered record is filed. After being POSTed, EYMSP030 returns to EYMSP040 for record processing. EYMSP030 will be re-entered when processing has been completed to WAIT on the next event.

**INPUT** Address of the Common Program Storage. Status of the PostProcessor

**OUTPUT** None

**Return codes**
- 0 - no errors
- 4 - warning
- 8 - error
- 12 - severe error
- 16 - catastrophic error

## EYMSP040

**MODULE NAME**

EYMSP040

**FUNCTION** EYMSP040 Tivoli Information Management for z/OS Link Module

**OVERVIEW** EYMSP040 Tivoli Information Management for z/OS Link Module.
- Invoked as Tivoli Information Management for z/OS exit
- LINK to EYMSP030 for wait
- Execute Tivoli Information Management for z/OS PostProcess after EYMSP030 is POSTed
- RETURN to EYMSP020 for termination (via Tivoli Information Management for z/OS/TSO)

**DESCRIPTION**

EYMSP040 is the interface between the Tivoli Information Management for z/OS process and the MVS environment.

**INPUT** None

**OUTPUT**

- TSCAFRET 0 = no errors
- TSCAFRET 8 = error
- TSCAFRES 0 = normal, record(s) to be processed
- TSCAFRES 4 = terminate PostProcessor

**Return codes**

- 0 - no errors
- 4 - warning
- 8 - error
- 12 - severe error
- 16 - catastrophic error

## EYMSP041

**MODULE NAME**
EYMSP041

**FUNCTION** Allocate/initialize Post Processor Mapping Table (PPMT)

**OVERVIEW** EYMSP041 Allocate/initialize in-storage PostProcessor mapping table

- VDEFINES ISPF variable `&EYMPPMT` for mapping table address
- GETMAINs a block of storage for mapping table
- Initializes select fields in mapping table
- VPUTs PPMT address in ISPF variable `&EYMPPMT`

**DESCRIPTION**
EYMSP041 is called as a user exit from TSP EYM9MAIN to allocate storage for the PostProcessor Mapping Table (PPMT) and initialize select fields within the PPMT.

**INPUT** None

**OUTPUT**

- PPMT - Post Processor Mapping Table - modified
- TSCA - Terminal Simulator Communications Area - modified

**Return codes**

- 0 - no errors
- 16 - catastrophic error

## EYMSP042

**MODULE NAME**
EYMSP042

**FUNCTION** Initialize/Reset the Post Processor Mapping Table (PPMT)

**OVERVIEW** EYMSP042 Initialize/reset the in-storage PostProcessor mapping table

- Compares input mapping reference record id with id of last record processed
- If table is to be reused, it resets row flags
- If a new table is to be built, it initializes table header

**DESCRIPTION**
EYMSP042 is called as a user exit from TSP EYM9MAPB to either initialize or reset the in-storage mapping reference list (PPMT). If the mapping reference record passed via the TSCA is the same as the ID stored

in the PPMT, the PPMT table is retained with any row and table flags initialized. If the IDs are different, the rows of the PPMT are discarded.

**INPUT**

- TSCA - Terminal Simulator Communications Area
- PPMT - Post Processor Mapping Table

**OUTPUT** PPMT - Post Processor Mapping Table - modified

**Return codes**

- 0 - no errors
- 16 - VGET of PPMT address failed

## EYMSP043

**MODULE NAME**
EYMSP043

**FUNCTION** Load row from mapping reference record into storage

**OVERVIEW** EYMSP043 Loads row from mapping reference record into storage

- Parses Target Panel, Source Prefix and Target Field/Command out of variable data area
- Loads parsed values into Post Processor Mapping table

**DESCRIPTION**
EYMSP043 is called as a user exit from TSP EYM9MAPB to parse mapping reference record data out of the TSP variable data area and load it into the Post Processor Mapping Table, the in-storage representation of the mapping reference record.

**INPUT**

- TSCA - Terminal Simulator Communications Area
- PPMT - Post Processor Mapping Table
-

**OUTPUT** PPMT - Post Processor Mapping Table - modified

**Return codes**

- 0 - no errors
- 8 - data length errors were found
- 16 - VGET of PPMT address failed

## EYMSP044

**MODULE NAME**
EYMSP044

**FUNCTION** Gets next available PPMT row for current panel

**OVERVIEW** EYMSP044 Gets next available PPMT row for current panel

- Target Field/Command placed in TSP Variable data area
- Source Prefix placed in TSP user area
- Posts flag in TSP user area if row not found, or record filed

**DESCRIPTION**
EYMSP044 is called as a user exit from TSP EYM9MAPE to retrieve the next available row from the in-storage Post-Processor Mapping Table (PPMT). A flag is posted if a row is not found, or if a row is found indicating that the post-processed record should have been filed.

**INPUT**
- TSCA - Terminal Simulator Communications Area
- PPMT - Post Processor Mapping Table

**OUTPUT**
- TSCA - Terminal Simulator Communications Area - modified
- PPMT - Post Processor Mapping Table - modified

**Return codes**
- 0 - no errors
- 16 - VGET of PPMT address failed

## EYMSP045

**MODULE NAME**
EYMSP045

**FUNCTION** Initialize user areas in TSCA

**OVERVIEW** EYMSP045 Initializes user areas in TSCA
- Resets user field (TSCAUFLD) to blanks
- Set Variable Data Area length (TSCAVDAL) to zero

**DESCRIPTION**
EYMSP045 is called as a user exit from TSP EYM9MAPE to clear user areas in the TSCA prior to executing a command to post-process data. Clearing these areas assures that user TSPs are not corrupted by execution of the post-processor.

**INPUT** TSCA - Terminal Simulator Communications Area

**OUTPUT** TSCA - Terminal Simulator Communications Area - modified

**Return codes**
- 0 - no errors

## EYMSP050

**MODULE NAME**
EYMSP050

**FUNCTION** EYMSP050 PostProcessor Message Module

**OVERVIEW** EYMSP050 PostProcessor Message Handler Module.
- LINK to by EYMSP modules to issue a message

**DESCRIPTION**
EYMSP050 will issue a message when linked to by other EYMSP modules. The only exception will be for EYM000I; this message will be returned to EYMSP030 for a WTOR to be issued.

**INPUT** Parm list — Register 1 →
@ message number
@ insert #1
@ insert #2
@ X’80’ insert #n

**OUTPUT** The desired message is placed on the current message chain.

**Return codes**
- 0 - no errors

- 4 - warning
- 8 - error
- 12 - severe error
- 16 - catastrophic error

## EYMSP055

**MODULE NAME**
EYMSP055

**FUNCTION** Interface between TSP environment and exit EYMSP050

**OVERVIEW** EYMSP055 Bridges between TSP environment and message exit EYMSP050

- Obtains message id from TSCA
- Retrieves insert information when necessary
- Build parameter list and calls EYMSP050

**DESCRIPTION**
EYMSP055 is called to issue an error message from within the TSP environment. The message id is retrieved from the TSCA, Applicable inserts are retrieved, a parameter list is built, and a call is made to EYMSP050.

**INPUT**

- TSCA - Terminal Simulator Communications Area
- PPMT - PostProcessor Mapping Table

**OUTPUT** The desired message is placed on the current message chain.

**Return codes**

- 0 - no errors
- 16 - ISPF VGET of PPMT address failed

14

# NetView AutoBridge Messages

## Messages

This chapter lists and describes the messages issued by AutoBridge.

---

**EYL001I** *hh:mm:ss modname* **INPUT >** *parm1 parm2 parm3 parm4 parm5 parm6 parm7*

**Explanation:** The AutoBridge MOD trace was selected for one or more functions. This is the message produced whenever a module is entered.

**Destination:**

| | |
|---|---|
| *hh:mm:ss* | The current hour, minute, and second on this NetView system |
| *modname* | The name of the module being entered |
| *parm1*—*parm7* | The first seven parameters passed to this module |

**System Action:** This informational trace message is displayed, logged, or both depending on message suppression and automation table values.

**Operator Response:** None.

---

**EYL002I** *hh:mm:ss modname* **OUTPUT >** *exit_value*

**Explanation:** The AutoBridge MOD trace was selected for one or more functions. This is the message produced whenever a module is exited.

**Destination:**

*hh:mm:ss*
The current hour, minute, and second on this NetView system

*modname*
The name of the module being exited

*exit_value*
The return code or result specified on this module's exit or return point

**System Action:** This informational trace message is displayed, logged, or both depending on message suppression and automation table values.

**Operator Response:** None.

---

**EYL003I** *hh:mm:ss module* **DATA >** *d1 d2 d3 d4 d5 d6 d7*

**Explanation:** The AutoBridge DATA trace was selected for one or more functions. This is the message produced when certain modules produce data. For example, if tracing the AutoBridge API data, these messages contain the record being passed to the checkpoint task. If tracing the process table, these messages contain the ADD_DATA and PARSE results.

**Destination:**

*hh:mm:ss*
The current hour, minute, and second on this NetView system

*module* The name of the module producing the data

*d1—d7* The data fields (up to seven data fields per message line)

**System Action:** This informational trace message is displayed, logged, or both depending on message suppression and automation table values.

**Operator Response:** None.

---

**EYL004W** **AUTOBRIDGE EXEC** *modname* **CANCELED PER OPERATOR REQUEST**

**Explanation:** You entered either a RESET or a CANCEL command while an AutoBridge module was running.

**Destination:**

*modname*
The name of the module that was canceled

**System Action:** Processing for the indicated command stops.

**Operator Response:** None.

---

**EYL005W** **AUTOBRIDGE EXEC** *modname* **TIMED OUT**

**Explanation:** An AutoBridge module issued a command and was waiting for the expected response. For example, starting a task or loading a process table should produce a message. No message was received in either the time specified by the value of WAITTIME in the initialization table (EYLATINT) or, if no value was specified, the default timeout of five seconds.

**Destination:**

*modname*
The name of the module that encountered the timeout condition

**System Action:** Processing for the indicated module stops.

**Operator Response:** Notify your system programmer.

**Operator Response:** Look for the NetView or system log to determine which module failed and any associated error messages. If you are unable to solve the problem, increase the value of WAITTIME in the initialization table (EYLATINT).

---

**EYL006E** **AUTOBRIDGE EXEC** *modname* **FAILED — NO VALUE AT LINE** *linenum*

**Explanation:** An AutoBridge REXX module NOVALUE signal was raised.

**Destination:**

*modname*
The name of the module that encountered the “no value” condition

*linenum* The line number of the failing instruction

**System Action:** Processing for the indicated module stops.

**Operator Response:** Notify your system programmer.

**Operator Response:** Look for the NetView or system log to determine which module failed and any associated error messages. If unable to determine the cause or perform a correction, contact Tivoli Customer Support for additional programming assistance.

---

**EYL007E** **AUTOBRIDGE EXEC** *modname* **FAILED — SYNTAX ERROR** *error* **AT LINE** *linenum*

**Explanation:** An AutoBridge REXX module SYNTAX signal was raised.

**Destination:**

*modname*
The name of the module that encountered the "no value" condition.

*error* The type of syntax error encountered. These are documented in the *Procedures Language MVS/REXX Reference* manual.

*linenum* The line number of the failing instruction.

**System Action:** Processing for the indicated module stops.

**Operator Response:** Notify your system programmer.

**Operator Response:** Look for the NetView or system log to determine which module failed and any associated error messages. If unable to determine the cause or perform a correction, contact Tivoli Customer Support for additional programming assistance.

**EYL051E** **AUTOBRIDGE API INVOKED WITH INVALID PARAMETERS** *parm1 parm2 parm3*

**Explanation:** The invocation of the NetView AutoBridge API (ABAPI) did not include a valid process segment name, bridge dispatcher name and description, or label for input data.

**Destination:**

*parm1* The first parameter

*parm2* The second parameter

*parm3* The third parameter

**System Action:** No NetView AutoBridge processing occurs.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Verify that the NetView AutoBridge was invoked correctly.

**EYL052E** **THERE IS NO VALID TRANSACTION DATA FOR THE AUTOBRIDGE**

**Explanation:** The NetView AutoBridge API was invoked, but when the process table segment was complete, no *parmvar* data was created to send to the target database.

**System Action:** No NetView AutoBridge processing occurs.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Verify that the process table segment includes statements that will produce transaction data. You may choose to trace AutoBridge components.

**EYL053E** **AUTOBRIDGE STOPPED PROCESSING RECORD, RC=***rc* **FROM EXEC=***exec*

**Explanation:** The NetView AutoBridge API invoked an exec which produced a return code greater than 8 to stop processing this data.

**Destination:**

*rc* The return code

*exec* The exec name (command, command processor or command list)

**System Action:** No further NetView AutoBridge processing occurs.

**Operator Response:** None.

**System Programmer Response:** None required (assuming that this was a valid return code and the desired result). If this message was received unexpectedly, examine the invoked EXEC to determine the cause of the return code.

**EYL054E** **AUTOBRIDGE ENCOUNTERED INVALID COMMAND** *command*

**Explanation:** The AutoBridge API encountered an invalid command in the process table.

**Destination:**

*command*
The command as listed in the process table

**System Action:** The AutoBridge ignores this command and proceeds with the next process table statement.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Examine the process table and correct the statement.

**EYL055I** **AUTOBRIDGE API IS NOT ACTIVE. NO PROCESSING WILL OCCUR**

**Explanation:** The AutoBridge API was invoked but it is not in an active state.

**System Action:** The AutoBridge does not process the MSUSEG, message, or data buffer passed to it.

**Operator Response:** If the API was purposely stopped or the AutoBridge application was purposely not started, no action is required. If the API should be active, you may start the application by issuing ABRIDGE START ALL or ABRIDGE START API or by using the ABMENU screen.

**System Programmer Response:** If this was an unexpected state, review the system and NetView logs to determine why the API was inactive.

**EYL056E** **AUTOBRIDGE API UNABLE TO READ THE PROCESS TABLE** *process_segment*

**Explanation:** The process segment specified as the first parameter in the AutoBridge call cannot be accessed.

**Destination:**

*process_segment* The process segment that cannot be accessed

**System Action:** The AutoBridge does not process the MSUSEG, message, or data buffer passed to it.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Verify that the process segment name exists in the process table EYLATPRO. If the segment was added since the last time AutoBridge was started, use the ABTABLES process table load (option 10) to reload the process table.

**EYL150E** **THE PROCESS TABLE FUNCTION** *function* **WAS INVOKED AND A REQUIRED PARAMETER** *parm* **WAS MISSING**

**Explanation:** A required keyword was not found or a keyword was not valid for the invocation of a process table function.

**Destination:**

| | |
|---|---|
| *function* | The name of the process table function that was invoked |
| *parm* | The parameter that was not found |

**System Action:** The alias name associated with that function is deleted.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error, then reload the process table.

**EYL151E** **EXEC** *exec* **ENDED. RETURN CODE** *rc*

**Explanation:** An application or user-written exec returned a non-zero return code

**Destination:**

| | |
|---|---|
| *exec* | The name of the exec that returned a non-zero code |
| *rc* | The return code value |

**System Action:** The alias name associated with that function is deleted.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error, then retry.

**EYL152I** **MAPPING STATEMENT** *statement_details*

**Explanation:** This message precedes most error messages and identifies the statement that was being processed when an error was detected.

**Destination:**

| | |
|---|---|
| *statement_details* | The statement from the mapping segment |

**System Action:** None.

**Operator Response:** Give details of the statement to your system programmer.

**System Programmer Response:** Use this message and its related messages to identify the statement and segment where the failure occurred.

**EYL153E** **THE PROCESS TABLE FUNCTION** *function* **WAS TERMINATED DUE TO A SEGMENT OR TABLE ERROR**

**Explanation:** The process table function was not executed because the mapping segment could not be retrieved or because the table was not active.

**Destination:**

| | |
|---|---|
| *function* | The name of the process table function that was invoked |

**System Action:** Control returns to the AutoBridge API.

**Operator Response:** Verify that the mapping table is active. If the table is inactive, start the table. Notify your system programmer if the table had been started when the error occurred.

**System Programmer Response:** Correct the problem, then reload the mapping table.

**EYL154E** **THE PARAMETER** *parm* **IS NOT VALID**

**Explanation:** An invalid parameter was passed to a process table function.

**Destination:**

| | |
|---|---|
| *parm* | The name of the parameter that failed |

**System Action:** Processing for the command stops.

**Operator Response:** If the command was entered from the command line, then check the syntax and re-enter. If the syntax is correct, contact your system programmer.

**System Programmer Response:** Verify and correct the command, then retry.

**EYL155W** **THE CODEPOINT *code* FOR BLOCKID *bkid* WAS NOT FOUND**

**Explanation:** The DECODE keyword was specified on a statement, but the specified codepoint and block ID could not be retrieved.

**Destination:**

*code* The codepoint value extracted from the input data

*bkid* The block ID value extracted from the input data

**System Action:** The decode operation stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Contact your service representative for assistance in diagnosing the cause of the failure.

**EYL156W** **THE CODEPOINT *codepoint* FOR SUBVECTOR *subvector* WAS NOT FOUND**

**Explanation:** DECODE was specified on the mapping statement but the codepoint could not be resolved to descriptive text. See also EYL152I.

**Destination:**

*codepoint*
The codepoint displayed in hex

*subvector*
The subvector table that was searched

**System Action:** The decode operation ends.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Verify that the correct input was used, then retry the operation.

**EYL157W** **THE DECODE OF *codepoint* FAILED. THE SUBVECTOR AND/OR CODEPOINT WAS NOT FOUND**

**Explanation:** The mapping statement specifies “decode”, but this could not be performed because the codepoint or subvector was missing. See also EYL152I.

**Destination:**

*codepoint*
The codepoint displayed in hex

**System Action:** Processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and reload the mapping table.

**EYL159W** **NO DATA ASSIGNED TO ALIAS *alias* FROM *d1 d2 d3 d4 d5 d6 d7 d8* — NONE FOUND OR WAS INVALID**

**Explanation:** The *from_field* specified on a mapping or ADD_DATA statement did not contain data. Consequently, the alias variable was not created or modified.

**Destination:**

*alias* The name of the alias variable

*d1—d8* The data fields (up to eight data fields per message line)

**System Action:** Processing of the statement stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Investigate the cause of the failure and, if necessary, correct the problem.

**EYL162E** **THE SEGMENT *segment* WAS NOT FOUND IN THE MAPPING TABLE**

**Explanation:** The table manager did not retrieve the named segment. The segment may not exist or may be marked unusable due to syntax errors.

**Destination:**

*segment* The name of the mapping table segment

**System Action:** Processing of the segment stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and reload the mapping table. Note that EYL152I identifies the failing statement.

**EYL163E** **PROCESSING TERMINATED FOR SEGMENT *segment*. INPUT DATA NOT FOUND**

**Explanation:** The process table was invoked but the source data could not be found.

**Destination:**

*segment* The name of the process table segment

**System Action:** Processing of the segment stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and reload the mapping table. Note that EYL152I identifies the failing statement.

**EYL167I** **THE INPUT DID NOT PASS THE AUTOBRIDGE FILTER. PROCESSING IS BEING TERMINATED**

**Explanation:** The AutoBridge filtering was invoked and the input failed the filtering criteria. Processing of the input data stops and the checkpoint record is discarded.

**System Action:** Processing of the input stops.

**Operator Response:** None.

**System Programmer Response:** None required (assuming that this transaction should have been filtered).

**EYL200E** **THE ABRIDGE COMMAND MUST BE ISSUED WITH START, STOP OR RECYCLE**

**Explanation:** The ABRIDGE command was issued but the keyword of START, STOP, or RECYCLE was not entered as the action value.

**System Action:** AutoBridge start/resume activity stops.

**Operator Response:** If ABRIDGE was entered as a direct command, reissue the command with the proper syntax. If ABRIDGE was issued from a NetView automation table or via the ABMENU panel, notify your system programmer.

**System Programmer Response:** If ABRIDGE was issued from the NetView automation table, correct the syntax and reload the NetView automation table. If ABRIDGE was issued via the ABMENU panel, contact Tivoli Customer Support for additional programming assistance.

**Note:** The correct syntax is EYLEHBRG START|STOP|RECYCLE *component*.

**EYL201E** **THE ABRIDGE COMMAND MUST BE ISSUED WITH ALL|API|DISP|ADPT**

**Explanation:** The ABRIDGE command was issued but the keyword of ALL, API, DISP, or ADPT was not entered as the component value.

**System Action:** AutoBridge start/resume activity stops.

**Operator Response:** If ABRIDGE was entered as a direct command, reissue the command with the proper syntax. If ABRIDGE was issued from a NetView automation table or via the ABMENU panel, notify your system programmer.

**System Programmer Response:** If ABRIDGE was issued from the NetView automation table, correct the syntax and reload the NetView automation table. If ABRIDGE was issued via the ABMENU panel, contact Tivoli Customer Support for additional programming assistance.

**Note:** The correct syntax is ABRIDGE *action* ALL|API|DISP|ADPT.

**EYL202E** **ENDING *segment1* BUT ACTIVE SEGMENT IS *segment2* in EYLATINT MEMBER**

**Explanation:** The EYLATINT member (initialization data) contains an END statement that does not match the previous BEGIN statement. Proper initialization values cannot be created.

**Destination:**

*segment1*
: Label on the END statement

*segment2*
: Label on the previous BEGIN statement

**System Action:** AutoBridge start/resume activity stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Edit the EYLATINT member to contain matching BEGIN/END pairs and reissue the ABRIDGE command.

**Note:** See “Step 5. Plan the Initialization Table” on page 138 for the layout of the initialization table.

**EYL203E** **FAILED TO LOAD EYLATINT INITIALIZATION MEMBER**

**Explanation:** The EYLATINT member (initialization data) could not be loaded.

**System Action:** AutoBridge start/resume activity stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Contact Tivoli Customer Support to report an error in module EYLSTMEM.

**EYL204E** **INITIALIZATION MEMBER EYLATINT NOT FOUND IN DSIPARM DATASET(S)**

**Explanation:** The EYLATINT member (initialization data) was not located in any of the defined DSIPARM data sets.

**System Action:** AutoBridge start/resume activity stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Either create or copy the EYLATINT member to a concatenated DSIPARM data set or include the new DSIPARM data set in the NetView procedure, recycle NetView, and reissue the ABRIDGE command.

**EYL205W** **CANNOT START TABLE MANAGER TASK, ALREADY STARTED**

**Explanation:** A request was made to START ALL of the ABRIDGE components. This automatically includes the table manager and checkpoint manager subtasks. A started subtask cannot be started; you must RECYCLE or STOP, then START.

**System Action:** None.

**Operator Response:** If ABRIDGE was entered as a direct command or via the ABMENU panel, RECYCLE or STOP then START ALL to restart the subtasks. If ABRIDGE was issued from a NetView automation table, notify your system programmer.

**System Programmer Response:** If ABRIDGE was issued from the NetView automation table, correct the syntax and reload the NetView automation table.

**EYL206E** **TABLE MANAGER TASK FAILED TO START**

**Explanation:** A request was made to start all of the ABRIDGE components. The START ALL command includes the table manager and checkpoint manager subtasks. The START command failed.

**System Action:** The remainder of the AutoBridge START/RECYCLE activity continues.

**Operator Response:** Browse the NetView or system log for more detailed messages regarding this failure. If you are unable to resolve the problem, contact your system programmer.

**System Programmer Response:** Browse the NetView or system log for more detailed messages regarding this failure. Correct the error or contact Tivoli Customer Support for additional programming assistance.

**EYL207W** **CANNOT START CHECKPOINT VSAM, ALREADY STARTED**

**Explanation:** A request was made to START ALL of the ABRIDGE components. The START ALL command includes the table manager and checkpoint manager subtasks. A started subtask cannot be started; you must RECYCLE or STOP, then START.

**System Action:** None.

**Operator Response:** If ABRIDGE was entered as a direct command or via the ABMENU panel, RECYCLE or STOP then START ALL to restart the subtasks. If ABRIDGE was issued from a NetView automation table, notify your system programmer.

**System Programmer Response:** If this command was issued from the NetView automation table, correct the syntax and reload the NetView automation table.

**EYL208E** **CHECKPOINT VSAM TASK FAILED TO START**

**Explanation:** A request was made to START ALL of the ABRIDGE components. The START ALL command includes the table manager and checkpoint manager subtasks. The START command failed.

**System Action:** The remainder of the AutoBridge START/RECYCLE activity continues.

**Operator Response:** Browse the NetView or system log for more detailed messages regarding this failure. If you are unable to resolve the problem, contact your system programmer.

**System Programmer Response:** Browse the NetView or system log for more detailed messages regarding this failure. Correct the error or contact Tivoli Customer Support for additional programming assistance.

**EYL209E** **BEGIN STATEMENT IN EYLATINT BUT NO SECTION LABEL SPECIFIED**

**Explanation:** A BEGIN statement was encountered in the initialization member (EYLATINT). The statement did not include the required section label such as BEGIN XYZ.

**System Action:** AutoBridge start/resume activity stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Edit the EYLATINT member so that it contains matching BEGIN/END pairs and reissue the EYLEHBRG command.

**Note:** See "Step 5. Plan the Initialization Table" on page 138 for the layout of the initialization table.

**EYL210E** **AUTOBRIDGE TASK** ***task*** **FAILED TO START IN ALLOTTED TIME**

**Explanation:** An AUTOTASK command was issued for this dispatcher or checkpoint task based on a START or RECYCLE command and the expected response was not received in the allotted time. The task may still be in the process of starting, or it may have experienced a permanent error.

**Destination:**

| | |
|---|---|
| *task* | The dispatcher or checkpoint task name |

**System Action:** The remainder of the AutoBridge START/RECYCLE activity continues.

**Operator Response:** Browse the NetView or System Log for more detailed messages regarding this failure. If necessary, contact the system programmer.

**System Programmer Response:** Browse the NetView or System Log for more detailed messages regarding this failure. If the expected response was logged in the MVS system log (message DSI020I), you may want to increase the WAITTIME value in the EYLATINT initialization table. If a permanent error occurred, such as an invalid name specified in the initialization table, correct the table and retry. You may contact Tivoli Customer Support for additional programming assistance.

**Note:** This may be followed by message EYL214I because this is not considered a fatal error.

**EYL211W** **CANNOT START AUTOBRIDGE TASK** ***task*****, ALREADY STARTED**

**Explanation:** A request was made to START a dispatcher or checkpoint task that is already started. You must RECYCLE or STOP then START.

**Destination:**

| | |
|---|---|
| *task* | The dispatcher or checkpoint task name |

**System Action:** The remainder of the AutoBridge START activity continues.

**Operator Response:** If this was entered as a direct command or via the EYLMENU panel, RECYCLE or STOP then START ALL to restart the dispatcher. If this was issued from a NetView automation table, notify your system programmer.

**System Programmer Response:** If this command was issued from the NetView automation table, you may have to change this command to RECYCLE if it is possible that the dispatchers are already started.

**EYL212E** **ADAPTER** ***adapter*** **FAILED TO** ***action*** **IN THE ALLOTTED TIME**

**Explanation:** An MVS START or MVS STOP command was issued for this adapter based on a START, RECYCLE or STOP command, and the expected response was not received in the allotted time. The adapter may still be in the process of starting or stopping, or it may have experienced a permanent error.

**Destination:**

| | |
|---|---|
| *adapter* | The adapter name |
| *action* | The action taken (START, STOP, or RECYCLE) |

**System Action:** The remainder of the AutoBridge START/RECYCLE/STOP activity continues.

**Operator Response:** Browse the NetView or System Log for more detailed messages regarding this failure. If necessary, contact the system programmer.

---

**EYL213W** **CANNOT START ADAPTER** ***adapter*****, ALREADY STARTED**

**Explanation:** A request was made to START an adapter that is already started. You must RECYCLE or STOP, then START.

**Destination:**

*adapter* The adapter name

**System Action:** The remainder of the AutoBridge START activity continues.

**Operator Response:** If ABRIDGE START was entered as a direct command or via the EYLMENU panel, RECYCLE or STOP then START ALL to restart the adapter. If ABRIDGE START was issued from a NetView automation table, notify your system programmer.

**System Programmer Response:** If ABRIDGE START command was issued from the NetView automation table, you may have to change this command to RECYCLE if it is possible that the adapters are already started.

---

**EYL214I** **AUTOBRIDGE** ***action*** **COMPLETED SUCCESSFULLY**

**Explanation:** The requested action was completed with no severe errors.

**Destination:**

*action* The action taken (START, STOP, or RECYCLE)

**System Action:** None.

**Operator Response:** None.

**System Programmer Response:** None.

---

**EYL215E** **AUTOBRIDGE FAILED TO** ***action*** **SUCCESSFULLY**

**Explanation:** The requested action encountered severe errors.

**Destination:**

*action* The action that caused the errors (START, STOP, or RECYCLE)

**System Action:** The remainder of AutoBridge START/STOP/RECYCLE activity continues.

**Operator Response:** Browse the NetView or system log for more detailed messages regarding this failure. If you are unable to solve the problem, contact your system programmer.

**System Programmer Response:** Browse the NetView or system log for more detailed messages regarding this failure. Correct the error or contact Tivoli Customer Support for additional programming assistance.

---

**EYL216I** **AUTOBRIDGE** ***action*** **COMMAND IN PROCESS**

**Explanation:** The requested action is starting.

**Destination:**

*action* The action that is starting

**System Action:** Command processing continues.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL217E** **BEGIN LABEL *label* IN EYLATINT LONGER THAN 8 CHARACTERS**

**Explanation:** The label on a BEGIN/END segment in the initialization table (EYLATINT) is longer than the maximum of eight characters.

**Destination:**

*label* The label on the BEGIN database_segment or BEGIN record_type segment

**System Action:** The AutoBridge START or RECYCLE action ends.

**Operator Response:** Contact your system programmer.

**System Programmer Response:** Correct the initialization table so that all BEGIN database_segment and BEGIN record_type labels are eight characters or less. Also, correct any process list segment referring to those database_segment and record_type labels. Once corrected, reissue the ABRIDGE START or RECYCLE command.

**EYL250E** **MESSAGE ID *msgid* INVALID, MUST BE "NNN", "ABCNNN" OR "ABCDNNN"**

**Explanation:** The message ID used on the EYLSMSG command was specified in an incorrect format.

**Destination:**

*msgid* The message identifier specified on the call to EYLSMSG

**System Action:** None.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** If the message originated within an application supplied by Tivoli, contact Tivoli Customer Support. Otherwise, determine the cause of the error and correct it.

**EYL251I** **MESSAGE ID NUMERIC *msgid* IS NOT NUMERIC**

**Explanation:** The specified message ID is not a numeric value.

**Destination:**

*msgid* The message number to be displayed

**System Action:** None.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** If the message originated within an application supplied by Tivoli, contact Tivoli Customer Support. Otherwise, determine the cause of the error and correct it.

**EYL252I** **TOO FEW PARMS ON EYLSMSG COMMAND, 2 IS MINIMUM**

**Explanation:** The EYLSMSG command processor was called without the two required parameters.

**System Action:** None.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** If the message originated within an application supplied by Tivoli, contact Tivoli Customer Support. Otherwise, determine the cause of the error and correct it.

**EYL400I** **TABLE** *table* **WAS SUCCESSFULLY LOADED AT** *time* **ON** *date* **BY** *operid*

**Explanation:** The specified table loaded correctly.

**Destination:**

| | |
|---|---|
| *table* | The name of the table |
| *time* | The time of the load |
| *date* | The date of the load |
| *operid* | The name of the operator who initiated the load |

**System Action:** Processing continues and the table is available for use.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL401W** **SEGMENT NAME** *segment* **ON BEGIN STATEMENT DOES NOT MATCH NAME ON END STATEMENT**

**Explanation:** The segment name on the BEGIN statement does not match the name on the corresponding END statement. This message is preceded by message EYL424I.

**Destination:**

| | |
|---|---|
| *segment* | The segment name on the BEGIN statement |

**System Action:** The segment is marked as unavailable for further processing.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the cause of the failures, then reload the table.

**EYL403W** **THE NAME** *name* **NOT ALLOWED FOR AUTOBRIDGE** *action*

**Explanation:** The load request was entered for a table name that is not allowed by AutoBridge. This message is preceded by message EYL424I.

**Destination:**

| | |
|---|---|
| *name* | The name of the table |
| *action* | The table action (LOAD, DISPLAY,STATUS) being requested |

**System Action:** The load operation stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the table name, then retry the operation.

**EYL404E** **INVALID OPTIONS SPECIFIED ON** *action* **REQUEST**

**Explanation:** An invalid option was detected on the EYLSTMGR command. This message is preceded by message EYL424I.

**Destination:**

| | |
|---|---|
| *action* | LOAD, STATUS, TEST, or DISPLAY |

**System Action:** The load operation stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the command then retry the operation.

**EYL405I** **TEST COMPLETE. NO ERRORS DETECTED IN** *member*

**Explanation:** The test of the DSIPARM member was successful with no errors detected.

**Destination:**

*member* The name of the DSIPARM member

**System Action:** None.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL406E** **TEST COMPLETE. ERRORS DETECTED IN** *member*

**Explanation:** The test of the DSIPARM member detected errors in the member.

**Destination:**

*member* The name of the DSIPARM member

**System Action:** Operation stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the errors, then retry the operation.

**EYL407I** **TABLE** *table* **IS ACTIVE. LOADED AT** *time* **ON** *date* **BY** *operid*

**Explanation:** The message is in response to a status request on an AutoBridge table.

**Destination:**

| | |
|---|---|
| *table* | The name of the table |
| *time* | The time the table was loaded |
| *date* | The date the table was loaded |
| *operid* | The name of the operator who initiated the load |

**System Action:** None.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL408E** **REQUIRED PARAMETERS ON TABLE MANAGER COMMAND INVALID OR MISSING**

**Explanation:** The command was not constructed correctly; required parameters may be missing or misspelled. This message is preceded by message EYL424I.

**System Action:** Processing of the command stops.

**Operator Response:** Correct and re-enter the command, or contact your system programmer.

**System Programmer Response:** Correct the command, then retry.

**EYL409I** **TABLE** *table* **IS NOT LOADED**

**Explanation:** A request was made to a table that is not currently active.

**Destination:**

*table* The name of the table

**System Action:** None.

**Operator Response:** See “Managing the AutoBridge Tables” on page 125 for information on loading tables, or contact your system programmer.

**System Programmer Response:** Load the table and check for successful completion.

**EYL410W** **THE REQUESTED SEGMENT** ***segment*** **WAS NOT FOUND**

**Explanation:** The segment name specified on a display or retrieve request was not found in the specified table.

**Destination:**

*segment* The name of the segment specified on the command

**System Action:** The request is canceled.

**Operator Response:** Correct the command and re-enter it. If the message reappears, then contact your system programmer.

**System Programmer Response:** Verify the segment name and retry the operation.

**EYL411W** **THE TABLE MANAGER IS NOT ACTIVE**

**Explanation:** The table manager task is not active.

**System Action:** The requested operation cannot be started.

**Operator Response:** Start the table manager subtask.

**System Programmer Response:** None.

**EYL412W** **STATEMENT ERRORS. A DELIMITER MAY BE MISSING**

**Explanation:** One or more errors are associated with a statement in the mapping, process, or filter table. This message is preceded by message EYL424I.

**System Action:** The statement or segment that contains this statement will be ignored in future processing.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the errors and reload the table.

**EYL413W** **AN END STATEMENT WAS NOT FOUND FOR SEGMENT** *segment*

**Explanation:** A segment in a mapping table or process table does not have the required END statement.

**Destination:**

*segment* The name of the segment specified on the BEGIN statement

**System Action:** The segment will not be available for future processing.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the errors and reload the table.

**EYL414E** **TABLE** *table* **FAILED TO LOAD DUE TO EXCESSIVE ERRORS**

**Explanation:** All segments in a table contain one or more errors.

**Destination:**

*table* The name of the table

**System Action:** The load is ended and the previously loaded table is restored (if one existed).

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the errors and reload the table.

**EYL415W** **UNEXPECTED STATEMENT TERMINATION.**

**Explanation:** A semicolon was detected before all of the expected clauses on a statement were processed. This message is preceded by message EYL424I.

**System Action:** The statement and associated segment are marked as unavailable for further processing.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the errors and reload the table.

**EYL416W** **SEGMENT** *segment* **NOT USABLE DUE TO ERRORS**

**Explanation:** The segment name specified on a display or retrieve request has errors and has been marked as unavailable.

**Destination:**

*segment* The name of the segment specified on the command

**System Action:** The request is not processed.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the errors then reload the table.

**EYL417I** **START OF DATA FROM TABLE** *table*

**Explanation:** The message is part of a multi-line message and marks the start of the data retrieved from the specified table.

**Destination:**

*table* The name of the table

**System Action:** The request is successfully completed.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL418I** **DATA IS:** *dataline*

**Explanation:** This message is part of a multiline message and is generated for each statement of the table that is retrieved.

**Destination:**

*dataline* The statement from the specified table

**System Action:** Processing completes successfully.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL419I** **END OF DATA FROM TABLE** *table*

**Explanation:** This message is part of a multiline message and marks the end of the data retrieved from the table.

**Destination:**

*table* The name of the table

**System Action:** Processing completes successfully.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL421W** **A STATEMENT WAS FOUND BUT IT WAS NOT ASSOCIATED WITH ANY SEGMENT**

**Explanation:** A statement is not within a segment. The statement will not be accessible by the table manager. This message is preceded by message EYL424I.

**System Action:** Processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and reload the table.

**EYL422W** **A DELIMITER WAS NOT FOUND FOR THIS STATEMENT**

**Explanation:** A statement was encountered that does not end with a semicolon. This message is preceded by message EYL424I.

**System Action:** Processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and reload the table.

**EYL423W** **THE PARAMETER *parm* IS NOT VALID**

**Explanation:** A statement was encountered that contained an incorrect keyword parameter. This message is preceded by message EYL424I.

**Destination:**

*parm* The invalid parameter

**System Action:** Processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and reload the table.

**EYL424I** **ERROR DATA:** *user data*

**Explanation:** This message contains the data that was being processed when an error was detected.

**Destination:**

*user data*
The statement or parameter that was being processed when the error was detected

**System Action:** Processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and reload the table.

**EYL425E** **LOAD OF TABLE *table* HAS BEEN TERMINATED BECAUSE OF ERRORS**

**Explanation:** A storage failure occurred during load, or all table segments contain one or more errors.

**Destination:**

*table* The name of the table that was being processed when the failure occurred

**System Action:** AutoBridge processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and reload the table.

**EYL426E** **NO STORAGE AVAILABLE TO PROCESS COMMAND**

**Explanation:** Storage constraints resulted in an AutoBridge command failure.

**System Action:** AutoBridge processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and reload the table.

**EYL427W** **IMBEDDED 'BEGIN' STATEMENTS NOT ALLOWED**

**Explanation:** Another BEGIN statement is not allowed inside of a segment.

**System Action:** AutoBridge processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and reload the table.

**EYL428E** **PROGRAM NOT LOADED. STORAGE FAILURE OCCURRED**

**Explanation:** The table manager could not load one of its modules because of a storage failure.

**System Action:** AutoBridge processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Recycle the table manager.

**EYL430E** **TABLE** *table* **FAILED TO LOAD DUE TO READ ERRORS**

**Explanation:** The table could not be read, possibly because of allocation errors.

**Destination:**

*table* The name of the table that was being processed

**System Action:** AutoBridge processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and reload the table.

**EYL431E** **TABLE** *table* **NOT FOUND OR COULD NOT BE OPENED**

**Explanation:** The DSIPARM member was not found or the member could not be opened.

**Destination:**

*table* The name of the table that was being processed

**System Action:** AutoBridge processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and reload the table.

**EYL432E** **INVALID PARAMETER. VALID PARMS ARE: LOAD, DISPLAY, TEST, STATUS**

**Explanation:** The table manager was invoked with an invalid request.

**System Action:** Processing stops.

**Operator Response:** If the table manager command was entered from the NCCF command line, re-enter the command with a valid request. Otherwise, contact your system programmer.

**System Programmer Response:** See “Coding NetView AutoBridge Tables” on page 81 for the correct syntax of the table manager command and make the necessary corrections.

**EYL433E** **THE OPERATOR *operator* IS NOT SUPPORTED FOR THE FILTER TABLE**

**Explanation:** An invalid filter statement was encountered.

**Destination:**

*operator*
The unknown operator symbol

**System Action:** Processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** See "Coding the Filter Table" on page 97 for the correct syntax of a filter statement and make the necessary corrections.

**EYL435E** **A SEQUENCE ERROR IS DETECTED FOR THE OPERATOR:** *operator*

**Explanation:** A filter statement operator is being incorrectly used. For example, an EQUAL (=) was found where an AND (&) was expected.

**Destination:**

*operator*
The unknown operator symbol

**System Action:** Processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** See "Coding the Filter Table" on page 97 for the correct syntax of a filter statement and make the necessary corrections.

**EYL436E** **UNBALANCED PARENTHESIS ENCOUNTERED IN STATEMENT**

**Explanation:** The number of right parentheses does not match the number of left parentheses in a given statement.

**System Action:** Processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Review the statement in error and ensure that all open parentheses are properly closed. Reload the AutoBridge table.

**EYL437E** **SEGMENT NAME *segment* IS TOO LONG**

**Explanation:** The specified table segment name exceeds the maximum allowable length of 32 characters.

**Destination:**

*segment* The name of the invalid segment

**System Action:** No further processing is performed on the segment.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the segment and reload the AutoBridge table.

**EYL450E** **SELECTION *sel* IS NOT DEFINED. ENTER A NUMBER BETWEEN 1 AND *maxvalue***

**Explanation:** You entered a selection that is not defined for the current menu.

**Destination:**

| | |
|---|---|
| *sel* | The selection you entered |
| *maxvalue* | The maximum selection number supported on this menu |

**System Action:** No AutoBridge processing occurs.

**Operator Response:** Enter a valid selection number to choose the desired option.

**EYL451E** **KEY *key* IS NOT DEFINED. TRY AGAIN**

**Explanation:** You pressed a function key that is not defined for the current menu.

**Destination:**

| | |
|---|---|
| *key* | The undefined function key |

**System Action:** No AutoBridge processing occurs.

**Operator Response:** Press a valid function key to choose the desired action.

**EYL452W** **CANNOT SCROLL BACKWARD. ALREADY AT TOP OF DATA**

**Explanation:** You pressed the backward function key (F7), but the screen is already displaying the top of the data.

**System Action:** No AutoBridge processing occurs.

**Operator Response:** Press a valid function key to choose the desired action.

**EYL453W** **CANNOT SCROLL FORWARD. ALREADY AT BOTTOM OF DATA**

**Explanation:** You pressed the forward function key (F8), but the screen is already displaying the bottom of the data.

**System Action:** No AutoBridge processing occurs.

**Operator Response:** Press a valid function key to choose the desired action.

**EYL454W** **CANNOT ACCESS CHECKPOINT FILE. AUTOBRIDGE IS NOT ACTIVE**

**Explanation:** You requested the “Manage the Checkpoint File” option but AutoBridge has not been activated. The Checkpoint VSAM and Autotask must be active in order to process the checkpoint file request.

**System Action:** No AutoBridge processing occurs.

**Operator Response:** You may start the AutoBridge application by entering “ABRIDGE” or by entering “ABMENU” to go to the AutoBridge menu and selecting option 1 (Start/Recycle/Stop), then selecting Action 1 (Start) at the “Perform Action on All” entry.

**EYL455W** **INVALID ENTRY FOR *level* TRACE OF *function*. TRACE VALUE UNCHANGED**

**Explanation:** You entered a value on the ABTRACE screen other than a forward slash (/) or blank (space or delete).

**Destination:**

| | |
|---|---|
| *level* | The level of tracing (ALL, MOD, DATA, or REXX) |
| *function* | The function for which tracing should be turned on or off (ALL, AUTOBRIDGE_API, CHECKPOINT_MANAGER, HIGH_LEVEL, PROCESS_TABLE, or TABLE_MANAGER) |

**System Action:** The original trace settings remain unchanged.

**Operator Response:** Enter the forward slash (/) for the tracing levels to turn on or blank out an entry by spacing over or deleting the slash for the tracing levels to turn off.

---

**EYL456W** **OPTS VALUE** *opts* **INVALID. ENTER S FOR SEARCH AND/OR U FOR UPDATE**

**Explanation:** You entered a value other than S or U in the Opts column of the checkpoint update panel.

**Destination:**

| | |
|---|---|
| *opts* | The value that was entered in the Opts column. |

**System Action:** The checkpoint update facility waits for a corrected entry, an END command, or a CANCEL command.

**Operator Response:** Correct the entry by entering either or both of the following, then retry:

| | |
|---|---|
| **S** | Specifies that this alias name should be included in the search list |
| **U** | Specifies that this alias name or text field should be included in the update list |

**System Programmer Response:** None.

---

**EYL457W** **CANNOT SEARCH OR UPDATE THIS TYPE OF PARMVAR DATA**

**Explanation:** You entered an S or a U in the Opts column next to an ASSOCDATA or VERIFIER parameter. You may specify search only on alias names; update only on alias names or text lines.

**System Action:** The checkpoint update facility waits for a corrected entry, an END command, or a CANCEL command.

**Operator Response:** Correct the entry and retry.

**System Programmer Response:** None.

---

**EYL458W** **CANNOT SEARCH TEXT PARMVAR DATA**

**Explanation:** You entered a U in the Opts column next to a text line parameter. You cannot search freeform text via the AutoBridge API.

**System Action:** The checkpoint update facility waits for a corrected entry, an END command, or a CANCEL command.

**Operator Response:** Correct the entry and retry.

**System Programmer Response:** None.

---

**EYL500W** *modname1* **COULD NOT EXECUTE** *modname2***, RC=***rc*

**Explanation:** The module specified in *modname1* received a non-zero return code while attempting to invoke the module specified in *modname2*.

**Destination:**

| | |
|---|---|
| *modname1* | The parent module that detected the error |
| *modname2* | The child module that was being invoked |
| *rc* | The return code acquired while attempting to invoke the child module |

**System Action:** If the return code is a positive number, it is from the called process. If the return code is a negative number, the called process is not invoked.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the failure and correct it.

**EYL501I** *modname* **EXECUTED WITH INVALID PARMLIST,** *parmlist*

**Explanation:** The command processor specified in *modname* was called with one of its required parameters missing.

**Destination:**

*modname*
The name of the module with missing parameters

*parmlist* The list of parameters passed when the module was invoked

**System Action:** Processing for the requested transaction stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the missing parameter and correct it.

**EYL502I** **TASK** *task* **INACTIVE, TRANSACTION NOT PROCESSED**

**Explanation:** A transaction request was submitted to a checkpoint manager task that is inactive.

**Destination:**

*task* The checkpoint autotask specified to handle the transaction request

**System Action:** Processing for the requested transaction stops.

**Operator Response:** Start the autotask.

**System Programmer Response:** None.

**EYL503W** **VARIABLE** *varname* **SPECIFIED AS PARMVAR COULD NOT BE ACCESSED**

**Explanation:** The EYLSCSUB command processor determined that the variable specified as the *parmvar* cannot be accessed in the calling routine's variable pool.

**Destination:**

*varname*
The variable name used as the *parmvar* variable

**System Action:** Processing for the requested transaction continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the inaccessible variable and correct it.

**EYL504E** **ERROR OCCURRED ACCESSING VARIABLE** *varname***, RC=** *rc*

**Explanation:** The module EYLSCSUB encountered an error other than "variable not initialized" while attempting to determine the value of the variable specified. The return code is that of the NetView CNMVARS macro.

**Destination:**

*varname*
The name of the variable that could not be determined by the EYLSCSUB routine

*rc* The return code as specified by the NetView CNMVARS macro

**System Action:** The request is not processed.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL505W** **VARIABLE** *varname* **SPECIFIED IN** *parmlist* **COULD NOT BE ACCESSED**

**Explanation:** The variable specified as *varname* in the parameter list *parmlist* could not be determined in the called routine's variable pool.

**Destination:**

*varname*
The name of the variable to be determined

*parmlist* The name of the parameter list containing the variable

**System Action:** Transaction processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL506E** **UNKNOWN TRANSACTION** *transid* **IN MODULE** *modname*

**Explanation:** The module invoked to process the specified transaction ID could not determine the transaction to be performed.

**Destination:**

*transid* The transaction requested

*modname*
The module that was invoked to process the transaction specified

**System Action:** Checkpoint-manager processing of this transaction is canceled.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL507E** **UNKNOWN FUNCTION SPECIFIED IN MODULE** *modname*

**Explanation:** The module specified by *modname* could not determine the function to be performed (GET, PUT, LIST, UPDATE, or DELETE).

**Destination:**

*modname*
The name of the module detecting the error

**System Action:** Processing of the requested transaction stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL508E** *modname* **FAILED TO ACQUIRE STORAGE, RC=** *rc*

**Explanation:** The module specified as *modname* could not acquire a block of storage needed to build a node in the NetView Bridge link list interface.

**Destination:**

*modname*
The name of the module detecting the error

*rc* The return code from the NetView CNMNAMS command

**System Action:** Transaction processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL509E** *modname* **COULD NOT EXPOSE GLOBAL VARIABLE** *varname*, **RC=** *rc*

**Explanation:** The module specified in *modname* detected an error while attempting to retrieve the global variable specified in *varname*.

**Destination:**

| | |
|---|---|
| *modname* | The name of the module that detected the error |
| *varname* | The name of the global variable to be exposed |
| *rc* | The return code of the CNMVARS command |

**System Action:** Transaction processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the missing global variable and correct it.

**EYL520I** *corrid* **SUCCESSFULLY COMPLETED BY CHECKPOINT MANAGER** *checkpoint_task*

**Explanation:** This message is written to the NetView log every time a transaction is successfully completed by a checkpoint task.

**Destination:**

| | |
|---|---|
| *corrid* | The correlation ID assigned to this transaction |
| *checkpoint_task* | The task ID of the checkpoint autotask that processed this transaction |

**System Action:** None.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL522E** **DATA RETURNED FROM VSAM DST** *task* **NOT CORRECT**

**Explanation:** The DST that manages the checkpoint VSAM file returned a buffer that does not have the expected structure.

**Destination:**

| | |
|---|---|
| *task* | The ID of the DST that returned the buffer |

**System Action:** Transaction processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Verify the VSAM checkpoint file data structure. Contact Tivoli Customer Support for additional programming assistance.

**EYL523W** **WARNING** *code* **DETECTED ON** *transid corrid* **REASONCODE** *rc*

**Explanation:** The checkpoint manager received a warning return code during an attempt to submit a transaction to the target database.

**Destination:**

| | |
|---|---|
| *code* | The warning code generated |
| *transid* | The transaction being attempted |
| *corrid* | The correlation ID of the transaction |
| *rc* | The reason code returned to further define the warning |

**System Action:** The response is ignored.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Find the warning/reason code pair in the Tivoli Information Management for z/OS documentation to determine the error and correct it.

---

**EYL524E** **FAILURE DURING WAIT FOR RESPONSE FROM** *modname***, RC=** *rc*

**Explanation:** An error was encountered while waiting for a response from the module specified by *modname*.

**Destination:**

*modname*
The name of the module being waited for

*rc* The return code set at the time the error is encountered

**System Action:** Transaction processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the error and correct it.

---

**EYL525I** *cmd_proc* **FAILED RC=** *rc***, DATA FROM DST NOT RETURNED**

**Explanation:** A command processor that has requested data from the VSAM DST failed while attempting to fetch the returned data.

**Destination:**

*cmd_proc*
The name of the command processor that encountered the error

*rc* The return code from the CNMGETD command

**System Action:** Transaction processing is halted.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

---

**EYL526E** *modname* **FAILED DURING WAIT FOR DATA FROM DST, RC=** *rc*

**Explanation:** The command processor specified by *modname* detected an error while waiting for requested data from the VSAM DST.

**Destination:**

*modname*
The name of the command processor that encountered the error

*rc* The return code from the correlation ID request

**System Action:** Transaction processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

---

**EYL527E** *function* **CHECKPOINT FILE DATA TO EYLTVSM FAILED, RC=** *rc*

**Explanation:** A command processor performing the specified function determined that the VSAM DST failed to process the data.

**Destination:**

*function* The function requested of the DST (GET, PUT, LIST)

*rc* The return code from attempting to submit the request

**System Action:** Transaction processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Find the return code for the CNMSMSG macro in the *NetView Using PL/I and C* manual. A subset of the possible return codes are:

| | |
|---|---|
| **4** | Not invoked from an allowed installation exit. |
| **24** | Nonzero return code from DSIGET macro. (28) |
| **88** | smtext is too long. |
| **116** | Invalid message type. |
| **120** | Invalid destination type. |
| **124** | Conflict between message and destination type. |
| **216** | DSIWLS failure. Log was inactive. |
| **220** | DSIMQS failure. Task was inactive. |
| **1000** | Bad return code, X, from DSIMQS. (28) |

**EYL528I** **TRANSACTION COUNTER COULD NOT BE ACCESSED, RC=** *rc*

**Explanation:** The EYLSCSUB command processor could not access the variable used to retain the last correlation ID used.

**Destination:**

*rc* The return code from the CNMVARS command, documented in the *NetView Using PL/I and C* manual. A subset of the possible return codes are:

| | |
|---|---|
| **20** | cvname not found or value of cvname is null. |
| **40** | cvdatlen was too small. Data truncated. |
| **52** | Invalid cvfunc. |
| **88** | cvdatlen less than (<) 0 or cvdata greater than (>) 255. |
| **108** | Invalid cvname. |
| **156** | Invalid cvpool. |
| **160** | The storage pointed to by cvdata is not addressable. |
| **14000 + X** | Nonzero return code, X. See values for X below. |
| **x=4** | Invalid variable name. |
| **x=8** | Variable name already defined in dictionary. |
| **x=12** | Insufficient storage. |
| **x=20** | Value length limit was exceeded. |
| **x=28** | No command procedure related to current action. |
| **x=32** | Data was truncated. |

**System Action:** Transaction processing continues with the transaction counter reset to @@@.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL529I** **GLOBAL** *varname* **COULD NOT BE ACCESSED BY** *modname***, RC=** *rc*

**Explanation:** A module specified by *modname* failed to retrieve the value of an AutoBridge common global variable.

**Destination:**

*varname*
The name of the global being retrieved

*modname*
The name of the module attempting to retrieve the variable

*rc* The return code of the CNMVARS command. See message EYL528I for a subset of possible return codes.

**System Action:** Transaction processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL530E** **ERROR SUBMITTING** *corrid* **TO** *dbase* **DATABASE, RC=** *rc*

**Explanation:** The dispatcher encountered an error submitting the specified transaction to the NetView Bridge adapter.

**Destination:**

*corrid* The correlation ID assigned to the transaction

*dbase* The name of the database to which the transaction was submitted

*rc* The return code from the CNMSNDT macro documented in the *NetView Bridge Implementation* manual. A subset of the possible return codes are:

| | |
|---|---|
| **4** | Nonzero return code. |
| **24** | Storage allocation failure. |
| **44** | Deregistration unsuccessful. Issued from an exit. |
| **56** | Time-out value is not valid. |
| **88** | MDS_MU length is not valid. |
| **220** | DSI6DT task is inactive. |
| **288** | The transaction request generated exceeds 31K. |
| **292** | Invalid task type. The service routine can be invoked only under an OST or a PPT. |
| **296** | One of the transaction header parameters contained an invalid value. |
| **300** | A severe error condition was encountered when the service routine attempted to build the transaction request. |
| **400** | Data type is not valid. |
| **404** | DATA missing or not valid. |
| **408** | MS application cannot send to itself. |
| **416** | MS application is not registered. |
| **420** | Operations management served application is not registered. |
| **424** | UOW missing or not valid. |
| **428** | RTI missing or is invalid. |
| **432** | OAN missing or is invalid. |
| **436** | DAN missing or is invalid. |

| | |
|---|---|
| **440** | Origin application name invalid. |
| **444** | Destination network ID missing or is invalid. |
| **448** | Destination LU name missing or is invalid. |
| **452** | Destination application name missing or is invalid. |
| **456** | OII in RTI does not match TVBOPID. |
| **460** | Reply is invalid. |
| **464** | Bad MUTYPE given. |
| **468** | Bad SYNCH option. |
| **472** | User list is full. |
| **1000 + X** | MQS failed while sending transaction request to DESTTASK. X is the return code from DSIMQS. |
| **4000 + Y** | Nonzero return code, Y, from DSIPUSH macro. |
| **9000 + Y** | Reply command is invalid. Y is the return code from DSICES. |
| **22000 + n** | The *n*th parameter block in the link list pointed to by the stparms field contains an invalid prmnaml, prmleng value, or both. |

**System Action:** The transaction is written to the checkpoint file, but transmission to the database stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

---

**EYL531I** *corrid* **TO** *dbase* **DATABASE SUCCESSFULLY SUBMITTED**

**Explanation:** The dispatcher has successfully submitted the transaction specified to the NetView Bridge adapter.

**Destination:**

| | |
|---|---|
| *corrid* | The correlation ID assigned to the transaction |
| *dbase* | The name of the database the transaction is for |

**System Action:** Processing continues.

**Operator Response:** None.

**System Programmer Response:** None.

---

**EYL532E** **SYNTAX ERROR DETECTED IN MODULE** *modname***, PARMS=** *parm_list*

**Explanation:** The module specified detected a syntax error in its parameter list.

**Destination:**

| | |
|---|---|
| *modname* | The name of the module producing this message |
| *parm_list* | The list of parameters with which the module was executed |

**System Action:** Processing of the transaction stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the failure and correct it.

**EYL533E** **DELETE ID** *corrid* **REQUEST FAILED, RC=***rc*

**Explanation:** A delete request could not be completed by the VSAM DST.

**Destination:**

| | |
|---|---|
| *corrid* | The correlation ID to be deleted |
| *rc* | The return code from the CNMKIO command |

**System Action:** Processing of the transaction stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the failure and correct it.

**EYL534E** **FAILURE ATTEMPTING TO ACQUIRE CORRELATION ID LOCK, RC=** *rc*

**Explanation:** The EYLSCSUB command processor could not acquire the lock needed to generate a correlation ID.

**Destination:**

| | |
|---|---|
| *rc* | The return code from the CNMLOCK macro |

**System Action:** Transaction processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL535E** **PUT DIRECT FAILED FOR** *ckptask* **KEY** *corrid***, RC=** *rc*

**Explanation:** A create checkpoint file record failed.

**Destination:**

| | |
|---|---|
| *ckptask* | The checkpoint manager the transaction was processed by |
| *corrid* | The correlation ID to be deleted |
| *rc* | The return code from the CNMKIO command |

**System Action:** Processing of the transaction stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the failure and correct it.

**EYL536E** **PUT UPDATE FAILED FOR** *ckptask* **KEY** *corrid***, RC=** *rc*

**Explanation:** A checkpoint file record update failed.

**Destination:**

| | |
|---|---|
| *ckptask* | The checkpoint manager that processed the transaction |
| *corrid* | The correlation ID to be deleted |
| *rc* | The return code from the CNMKIO command |

**System Action:** Processing of the transaction stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the failure and correct it.

**EYL537E** **BAD REQUEST FROM** *ckptask* **PASSED TO DST IN TRANSACTION** *corrid*

**Explanation:** The EYLTVSM DST could not determine the function being requested.

**Destination:**

| | |
|---|---|
| *ckptask* | The task ID of the checkpoint autotask that submitted the request |
| *corrid* | The correlation ID of the transaction to be processed |

**System Action:** Transaction processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL538E** **GET_EQ FAILED FOR** *ckptask* **KEY** *corrid*, **RC=** *rc*

**Explanation:** The DST attempting to retrieve a record from the checkpoint file received an error other than “not found”.

**Destination:**

| | |
|---|---|
| *ckptask* | The checkpoint manager that processed the transaction |
| *corrid* | The correlation ID to be deleted |
| *rc* | The return code from the CNMKIO command |

**System Action:** Processing of the transaction stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the failure and correct it.

**EYL539E** **KEY** *corrid* **NOT FOUND FOR** *ckptask*

**Explanation:** The DST attempting to retrieve a record for the specified checkpoint task could not find the record.

**Destination:**

| | |
|---|---|
| *corrid* | The correlation ID to be retrieved |
| *ckptask* | The checkpoint manager that processed the transaction |

**System Action:** None.

**Operator Response:** List the checkpoint entries to determine the correct checkpoint and resubmit the transaction.

**System Programmer Response:** None.

**EYL540E** *transid* **FOR** *corrid* **CANNOT BE PROCESSED, FIELD** *fld* **MISSING**

**Explanation:** A required field for the transaction specified could not be filled.

**Destination:**

| | |
|---|---|
| *transid* | The transaction ID being processed |
| *corrid* | The correlation ID assigned to this transaction |
| *fld* | The name of the field in error |

**System Action:** Transaction processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL541E** **TRANSACTION *corrid* RETRY COUNT EXCEEDED**

**Explanation:** The specified transaction has been resent to the target database the maximum number of times with no response.

**Destination:**

*corrid* The correlation ID of the transaction

**System Action:** The transaction is flagged as “failed” in the checkpoint file.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL542E** **INPUT DATA WILL EXCEED BUFFER LIMIT OF 31000 BYTES**

**Explanation:** The transaction data generated via the mapping table or process table will result in a data buffer larger than the 31000 byte limit.

**System Action:** Transaction processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL543I** **UPDATE FOR TRANSACTION *corrid* CANCELLED, NO UPDATE DATA SPECIFIED**

**Explanation:** A conditional create transaction detected a duplicate record, but an update transaction could not take place because no update data was specified.

**Destination:**

*corrid* The correlation ID assigned to this transaction

**System Action:** Transaction processing is halted and the checkpoint file entry for this transaction is flagged as failed.

**Operator Response:** Add update data to the transaction or delete the entry from the checkpoint file.

**System Programmer Response:** None.

**EYL546W** **TRANSACTION COUNTER NOT SAVED TO VSAM DATABASE, RC=*rc***

**Explanation:** A failure occurred when the transaction counter was saved to disk. The transaction counter is saved to disk after each transaction initiation.

**Destination:**

*rc* The return code from the GLOBALV SAVE command

**System Action:** Transaction processing continues. When NetView is recycled, correlation IDs might be duplicated before the 64 000 unique names limit.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL547I** **DELETE REQUEST *corrid* WAS SUCCESSFUL**

**Explanation:** This message is generated when a “delete checkpoint file” request completes successfully.

**Destination:**

*corrid* The checkpoint file to be deleted

**System Action:** The checkpoint file record is deleted.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL551I** *corrid dfl ckptask transid dbase disp type*

**Explanation:** This is the control line of the multiline message returned to the requester of a GET function.

**Destination:**

| | |
|---|---|
| *corrid* | The value of the correlation ID created by this module |
| *dfl* | The description flag of this transaction record |
| *ckptask* | The checkpoint task that created this record |
| *transid* | The value of the current transaction ID for the specified record |
| *dbase* | The value of the *dbase* variable identified in the parameter list to this module |
| *disp* | The NetView dispatcher that processes this transaction |
| *type* | The value of the record type identified in the parameter list to this module |

**System Action:** None.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL553I** *data_tag* **IS** *key val*

**Explanation:** This is the data line of the checkpoint manager multiline message. It is repeated within the message to list all the data fields within the transaction data.

**Destination:**

| | |
|---|---|
| *data_tags* | The identifier of the *parmvar* part the data belongs to |
| *key* | The key or alias name of the data item |
| *val* | The value of the data to be used in the specified key when a transaction is created from this checkpoint record |

**System Action:** None.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL554I** **END OF DATA**

**Explanation:** This is the "end of message" line in the checkpoint manager multiline message.

**System Action:** None.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL556E** **CORRELATION ID NOT EXTRACTED FROM** *transid* **RESPONSE**

**Explanation:** The transaction response processor could not retrieve the correlation ID from the transaction response data returned by the NetView Bridge adapter.

**Destination:**

| | |
|---|---|
| *transid* | The transaction ID from the response |

**System Action:** Transaction results cannot be returned to the originating task, and the checkpoint file record cannot be updated as the correlation ID is the VSAM key to the record.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL557E** **CHECKPOINT FILE PROCESSING FAILED, DST RETURNED BUFFER** *data*

**Explanation:** The checkpoint file processor received a buffer from the VSAM DST containing a bad response.

**Destination:**

| | |
|---|---|
| *data* | The first 80 bytes of the buffer returned by the checkpoint file VSAM DST |

**System Action:** Checkpoint file processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL558E** **RESEND INTERVAL ERROR, DATE AND TIME** *date time*

**Explanation:** The checkpoint file processor failed to calculate the time variance between the time the transaction was last sent and the current time.

**Destination:**

| | |
|---|---|
| *date* | The date the transaction was last sent |
| *time* | The time the transaction was last sent |

**System Action:** Checkpoint file processing stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Determine the cause of the error and correct it.

**EYL559I** *corrid* **FOUND DUPLICATE RECORD** *recno* **- WILL UPDATE**

**Explanation:** The checkpoint manager found one or more records matching search arguments. The record with the highest RECORDID value will be updated. If the IBCREATE transaction was specified, only the process and mapping table statements containing the ',UPDATE' keyword are updated.

**Destination:**

| | |
|---|---|
| *corrid* | The correlation ID associated with the IBCREATE or IBUPDATE transaction. |
| *recno* | The record number that the update has been submitted against. |

**System Action:** The update request is submitted.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL560I** *corrid* **IBUPDATE FAILED - RECORD** *rnid* **NOT FOUND - WILL ATTEMPT IBCREATE**

**Explanation:** A conditional create located a record and attempted to update the record, but the record was not found when the update was processed. The update of the rnid will not occur, and another conditional create will be attempted.

**Destination:**

| | |
|---|---|
| *corrid* | The correlation ID of the transaction. |
| *rnid* | The record number that could not be found. |

**System Action:** The rnid value contained in the checkpoint file record for this *corrid* is removed and the transaction is changed to an IBCREATE. The IBCREATE will then be retried if RETRYNUM is not exceeded.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL561I** **LIST FUNCTION RESPONSE DATA**

**Explanation:** This is the control line of the multiline message created by the checkpoint manager in response to the user's request to list checkpoint file data.

**System Action:** None.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL563I** *corrid dfl cdate ctime scnt sdate stime opid transid dbase disp type*

**Explanation:** This is the data line within the EYL561I multi-line message. It is repeated under EYL561I to list each of the checkpoint file records to be listed.

**Destination:**

| | |
|---|---|
| *corrid* | The correlation ID assigned to the transaction specified in the record |
| *dfl* | The description flag of this transaction record |
| *cdate* | The create date of this transaction record |
| *ctime* | The create time of this transaction record |
| *scnt* | The number of times this transaction has been sent to the database |
| *sdate* | The date this transaction was last sent to the database |
| *stime* | The time this transaction was last sent to the database |
| *opid* | The operator task ID that created this transaction record |
| *transid* | The transaction ID assigned to the transaction specified in the record |
| *dbase* | The database that processed this transaction |
| *disp* | The NetView Bridge dispatcher that processes this transaction |
| *type* | The type of database record this transaction is to be processed against |

**System Action:** None.

**Operator Response:** None.

**System Programmer Response:** None.

**EYL565W** *trans corrid*, **RESPCODE:** *respcode* **REASONCODE** *reasoncode*. **RETRY NOT SCHEDULED**

**Explanation:** The response processor received an error condition in a transaction response.

**Destination:**

| | |
|---|---|
| *trans* | The transaction ID associated with the transaction |
| *corrid* | The correlation ID of the transaction that failed |
| *respcode* | The response code specifying the type of error detected in the transaction |
| *reasoncode* | The reason code for the failure detailing the error within the type |

**System Action:** The checkpoint file record for this transaction is flagged as a failed transaction.

**Operator Response:** Attempt to determine the cause of the error and correct it.

**System Programmer Response:** None.

**EYL567I** **TRANSACTION** *trans corrid* **COMPLETED SUCCESSFULLY**

**Explanation:** This is the control line of a multiline message returned to the originating task of a transaction when that transaction completes successfully.

**Destination:**

| | |
|---|---|
| *trans* | The transaction ID associated with the transaction |
| *corrid* | The correlation ID of the transaction |

**System Action:** The checkpoint file record for this transaction is deleted.

**Operator Response:** None.

**System Programmer Response:** None.

**EYM000I** **THE POSTPROCESSOR IS ACTIVE, REPLY END TO TERMINATE POSTPROCESSOR** *id*

**Explanation:** This message informs the operator that the named PostProcessor is active and can be stopped with a reply of "end".

**Destination:**

| | |
|---|---|
| *id* | The PostProcessor ID |

**System Action:** None.

**Operator Response:** Enter **END** to stop the named PostProcessor.

**System Programmer Response:** None.

**EYM001E** **THE** *macro* **MACRO HAS FAILED IN MODULE** *modname* **WITH A RETURN CODE OF** *rc*

**Explanation:** A system macro has ended with a non-zero return code.

**Destination:**

| | |
|---|---|
| *macro* | The name of the macro which failed |
| *modname* | The name of the module in which the macro failed |
| *rc* | The return code received from the macro |

**System Action:** The PostProcessor stops with a SNAP macro dump of program and related storage areas.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and rerun the program.

**EYM002E** **THE** *macro* **MACRO HAS FAILED IN MODULE** *modname*

**Explanation:** A system macro has failed.

**Destination:**

| | |
|---|---|
| *macro* | The name of the macro which failed |
| *modname* | The name of the module in which the macro failed |

**System Action:** The PostProcessor stops with a SNAP of program and related storage areas.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Correct the error and re-execute the program.

**EYM003E** **POSTPROCESSOR *id* HAS FAILED TO START**

**Explanation:** The PostProcessor task failed to POST EYMSP010 that it had completed initialization.

**Destination:**

| | |
|---|---|
| *id* | The name of the PostProcessor which failed |

**System Action:** Processing of API records continues without the PostProcessor for this adapter.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Examine the console log to determine the reason for the PostProcessor's failure to start.

**EYM004W** **POSTPROCESSOR TERMINATING DUE TO BRIDGE TERMINATION**

**Explanation:** The PostProcessor task has detected that its associated BRIDGE task has ended.

**System Action:** The PostProcessor stops.

**Operator Response:** None.

**System Programmer Response:** None.

**EYM005W** **THE MAXIMUM NUMBER OF POSTPROCESSORS ARE ACTIVE**

**Explanation:** The EYMSP010 module has detected that 16 PostProcessors are currently active.

**System Action:** Processing of API records continues without the PostProcessor for this adapter.

**Operator Response:** None.

**System Programmer Response:** None.

**EYM006E** **INVALID CALL TO MESSAGE HANDLER EYMSP050**

**Explanation:** The EYMSP050 module has detected an error in a call made to it.

**System Action:** A SNAP macro dump is created and processing continues.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Before contacting Tivoli Customer Support, be sure the following are true:
- The PostProcessor has been correctly installed.
- The PostProcessor logic has not been modified such that program exits are running incorrectly.

If the error persists after you have verified that the PostProcessor has been installed and customized correctly, contact Tivoli Customer Support for additional programming assistance.

**EYM007E** **AN INTERNAL LOGIC ERROR WAS DETECTED IN *modulename***

**Explanation:** An unexpected and incorrect condition was detected by the named PostProcessor module.

**Destination:**

| | |
|---|---|
| *modulename* | The name of the module in which the error was detected |

**System Action:** The PostProcessor stops.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Before contacting Tivoli Customer Support, be sure the following are true:
- The PostProcessor has been correctly installed.
- The PostProcessor logic has not been modified such that program exits are running incorrectly.

If the error persists after you have verified that the PostProcessor has been installed and customized correctly, contact Tivoli Customer Support for additional programming assistance.

**EYM100E** **DATA TOO LONG TO STORE IN PPMT; RECORD ID:** *record_id* **REASON CODE:** *rc*

**Explanation:** Exit EYMSP043 attempted to write an entry to the PostProcessor mapping table (PPMT) and found that the entry is either missing or too long for the table column.

**Destination:**

*record_id*
The name of the record which failed

*rc* The reason code for the failure:

| | |
|---|---|
| **0** | TSCA variable data area is empty. |
| **1** | Error occurred loading Target Panel Name. |
| **2** | Error occurred loading Target Field/Command Name. |
| **3** | Error occurred loading Source Prefix Name. |

**System Action:** The PostProcessor stops with a SNAP macro dump of program and related storage areas.

**Operator Response:** None.

**System Programmer Response:** This error will occur if one of the following is true:

- The mapping reference record panels have been modified to accept data that is greater in length than the panels shipped by Tivoli accept.
- Exit EYMSP043 is improperly used in a new or user-modified TSP.

Restore the PostProcessor panels supplied by Tivoli, then update the record which caused the failure.

**EYM101E** **RECORD FILE MESSAGE NOT FOUND**

**Explanation:** TSP EYM9MAPE read an entry from the PostProcessor mapping table (PPMT) which contains a Target Panel value, but no Target Field/Command value, indicating that the record being post-processed should have been filed. TSP EYM9MAPE tested for Tivoli Information Management for z/OS message BLG03058 (“record filed successfully”), but did not find it and issued this message.

**System Action:** The post-processing of the current record stops.

**Operator Response:** None.

**System Programmer Response:** This error will occur only if both of the following conditions are true:

- The mapping reference record contains an entry with a Target Panel value, but no Target Field/Command value.
- At PostProcessor execution time, the Tivoli Information Management for z/OS dialog does not return to this panel after the post-processed record has been filed.

The mapping reference record must be updated to ensure that the correct file-processing panel flow is represented in the mapping reference data list.

**EYM102E** **UNEXPECTED PANEL FLOW; CURRENT PANEL:** *panel_id*

**Explanation:** TSP EYM9MAPE could not find an available entry in the PostProcessor mapping table (PPMT) for the current panel.

**Destination:**

*panel_id*
The name of the panel that could not be located

**System Action:** The post-processing of the current record stops.

**Operator Response:** None.

**System Programmer Response:** This error is issued when PostProcessor execution causes flow to a panel for which there is no available entry in the PPMT.

The mapping reference record must be updated to ensure that the correct panel flow is represented in the mapping reference data list.

**EYM103E** **FIELD/COMMAND PROCESS ERROR; MAP ID:** *map_id***; ROW:** *row*

**Explanation:** TSP EYM9MAPE processed an entry from the PostProcessor mapping table (PPMT) that resulted in a non-zero return code. Message EYM104I follows this message, listing the contents of the failed row.

**Destination:**

*map_id* The ID of the mapping reference record used to post-process the record

*row* The number of the row in the mapping reference record that was being executed when the error occurred

**System Action:** The post-processing of the current record stops.

**Operator Response:** None.

**System Programmer Response:** This error is issued when post-processing causes a non-zero return code to be returned from the PROCESS control line in TSP EYM9MAPE. This can be caused by incorrect data in the post-processed record or an incorrect Target Field/Command or Source Prefix value in the mapping reference record.

Check the mapping reference record to ensure that the correct post-processing flow is specified. If this appears correct, check the data in the record undergoing post-processing.

**EYM104I** **PANEL:** *panel_id***; COMMAND:** *command* **PREFIX:** *prefix***; DATA:** *data*

**Explanation:** This message follows message EYM103E to state the contents of the PPMT row that triggered execution failure.

**Destination:**

*panel_id*
The value of the Target Panel field for this row

*command*
The value of the Target Field/Command field for this row

*prefix* The value of the Source Prefix field for this row

*data* The first 16 bytes of the record that is associated with this source prefix

**System Action:** The post-processing of the current record stops.

**Operator Response:** None.

**System Programmer Response:** This message is issued when post-processing causes a non-zero return code to be returned from the PROCESS control line in TSP EYM9MAPE. This can be caused by incorrect data in the post-processed record or an incorrect Target Field/Command or Source Prefix value in the mapping reference record.

Check the mapping reference record to ensure that the correct post-processing flow is specified. If the mapping reference record appears correct, check the data in the record undergoing post-processing.

**EYM105I** **POST PROCESSING OF RECORD** *new_record_id* **HAS COMPLETED. SOURCE RECORD** *old_record_id* **HAS BEEN DELETED**

**Explanation:** This message is issued whenever the post-processing of a given record has completed successfully.

**Destination:**

*new_record_id* The ID of the record which was created

*old_record_id* The ID of the record which was deleted

**System Action:** Post-processing continues.

**Operator Response:** None.

**System Programmer Response:** None.

---

**EYM106E** **POST PROCESSING OF RECORD** *record_id* **HAS FAILED**

**Explanation:** The post-processing of a given record is unable to continue for conditions stated in error messages that immediately precede this message.

**Destination:**

*record_id*
The ID of the record which failed

**System Action:** The post-processing of the current record is cancelled, and the record is flagged to prevent the PostProcessor from attempting to re-process the record.

**Operator Response:** None.

**System Programmer Response:** Examine the error messages immediately preceding this message to determine the error conditions and the corrective action.

---

**EYM107E** **MAPPING REFERENCE RECORD ID:** *record_id* **NOT FOUND**

**Explanation:** The PostProcessor could not find a mapping reference record in the database of the same name as that specified in the record created by AutoBridge.

**Destination:**

*record_id*
The ID of the record which could not be found

**System Action:** The post-processing of the current record is cancelled

**Operator Response:** None.

**System Programmer Response:** Ensure the alias tables used by AutoBridge contain the correct mapping reference record ID or create a mapping reference record with a name equal to the above record ID.

---

**EYM108E** **UNEXPECTED RESPONSE RECEIVED; RESPONSE WAS:** *response_code*

**Explanation:** This message is issued when a PostProcessor TSP receives an unexpected response.

**Destination:**

*response_code*

```
1x - EYM9MAIN (PostProcessor is terminated)
2x - EYM9POST (PostProcessor is terminated)
3x - EYM9MAPB (PostProcessor is terminated)
4x - EYM9MAPE (PostProcessor is terminated)
5x - EYM9XMIT (Processing continues)
```

**Operator Response:** None.

**System Programmer Response:** You can view the appropriate TSP to try to determine the source of the error. For example, message EYM108E reported a response code of ’32’. The TSP list above indicates that 3x responses are from EYM9MAPB. View EYM9MAPB to find where ’32’ is issued.

```
LABEL       NOTFOUND * DISPLAY OF MAP REF REC FAILED*
LABEL                ********************************
TESTFLOW    ERR32
```

If you can’t determine the problem, contact Tivoli Customer Support service after verifying that the PostProcessor has been correctly installed and logic has not been modified such that program exits are being incorrectly executed.

**EYM13901E** **Target Panel data is required**

**Explanation:** A value for the Target Panel field is required when data is present in either the Target Field Command or Source Prefix fields.

**System Action:** The mapping reference data list is positioned at the row where the error was detected.

**Operator Response:** Correct the line by specifying a value in the Target Panel field, or by deleting the line in question.

**System Programmer Response:** None.

**EYM13902E** **Mapping Reference Data List is too long**

**Explanation:** The mapping reference data list can contain a maximum of 100 rows. This error message is issued because more than 100 rows were found.

**System Action:** The mapping reference data list is displayed for operator action.

**Operator Response:** Delete rows in the mapping reference data list until there are no more than 100 rows present.

**System Programmer Response:** None.

**EYM13903E** **User Node Field is required if User ID is specified.**

**Explanation:** The User Node field is required whenever a value has been specified in the corresponding User ID field.

**System Action:** The mapping reference record cannot be filed until the error is corrected.

**Operator Response:** Either specify a value for the corresponding User Node field, or clear the User ID field.

**System Programmer Response:** None.

**EYM13904E** **Record not purged due to referencing records.**

**Explanation:** An attempt was made to delete a mapping reference record whose record ID is contained in other records.

**System Action:** The mapping reference record is not deleted.

**Operator Response:** The mapping reference record cannot be deleted until all references are removed. Process the referencing records to remove the ID of record you wish to delete.

**System Programmer Response:** None.

**EYM999E** **EYMSP050**

**Explanation:** The Message Handler module, EYMSP050, was unable to locate message EYM006E in the Message module.

**Operator Response:** Notify your system programmer.

**System Programmer Response:** Before contacting Tivoli Customer Support, be sure the following are true:
- The PostProcessor has been correctly installed.
- The PostProcessor logic has not been modified such that program exits are being incorrectly processed.

If the error persists after you have verified that the PostProcessor has been installed and customized correctly, contact Tivoli Customer Support for additional programming assistance.

# 15

# NetView AutoBridge Worksheets

Use the worksheets in this chapter to plan the content of the initialization table, process table, mapping table, and filter table for your AutoBridge installation.

You may choose to photocopy the worksheets in this chapter to facilitate your installation planning.

## Initialization Table Worksheet

*Table 33. Initialization table worksheet*

| Description | Values |
|---|---|
| Type of AutoBridge application, resident or remote | 1. AUTOBRIDGE = ________ |
| Date format generated for the checkpoint manager data | 2. DATEFORMAT = __ |
| Wait time (in whole seconds) for issued commands | 3. WAITTIME = ____ |
| Remote dispatcher on the resident NetView | 4. RDISPATCH = ____________ |
| Bridge dispatcher, checkpoint manager, and adapters segment | 5. Dispatcher name: ______________ |
| | 6. CHECKPT = _______________ |
| | 7. ADAPTER = _______________ |
| | 8. Dispatcher name: ______________ |
| | 9. CHECKPT = _______________ |
| | 10. ADAPTER = _______________ |
| | 11. Dispatcher name: ______________ |
| | 12. CHECKPT = _______________ |
| | 13. ADAPTER = _______________ |
| | 14. Dispatcher name: ______________ |
| | 15. CHECKPT = _______________ |
| | 16. ADAPTER = _______________ |
| Database group | 17. Database group name: ___________ |
| Type of target database | 18. DATABASE = INFOMGMT |
| Resident network ID | 19. BRGNETID = ____________ |
| Resident domain ID | 20. DOMAINID = ____________ |
| Separation character | 21. SEPARATOR = ______ |
| Correlation alias name | 22. CORRID = ____________ |
| Send transaction processor | 23. SEND = ____________ |
| Receive transaction processor | 24. RECEIVE = ____________ |
| Number of times to resend transaction | 25. RETRYNUM = ____________ |
| Time in seconds between retries | 26. RETRYINT = ____________ |
| Create privilege class (optional) | 27. CREPRIV = ____________ |
| Update privilege class (optional) | 28. UPDPRIV = ____________ |
| Inquiry privilege class (optional) | 29. INQPRIV = ____________ |

*Table 33. Initialization table worksheet (continued)*

| Description | Values |
|---|---|
| Record group | 30. Record group name: _________ |
| Alias table | 31. VOCAB = ____________ |
| Create PIDT | 32. IBCREATE = ____________ |
| Update PIDT | 33. IBUPDATE = _____________ |
| Search PIDT | 34. IBSEARCH = ____________ |
| Inquiry PIDT | 35. INQVIEW = ____________ |
| | 36. Record group name: _________ |
| | 37. VOCAB = ____________ |
| | 38. IBCREATE = ____________ |
| | 39. IBUPDATE = ____________ |
| | 40. IBSEARCH = ____________ |
| | 41. INQVIEW = ____________ |
| | 42. Record group name: _________ |
| | 43. VOCAB = ____________ |
| | 44. IBCREATE = ____________ |
| | 45. IBUPDATE = ____________ |
| | 46. IBSEARCH = ____________ |
| | 47. INQVIEW = ____________ |
| | 48. Record group name: _________ |
| | 49. VOCAB = ____________ |
| | 50. IBCREATE = _____________ |
| | 51. IBUPDATE = ____________ |
| | 52. IBSEARCH = ____________ |
| | 53. INQVIEW = ____________ |

## Process Table Planning Worksheet

*Table 34. Process table worksheet*

| Function | Mapping segment | (✓) FILTER | EXEC () | (✓) SEARCH | (✓) UPDATE | (✓) TEXT |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

## Mapping Table Planning Worksheet

*Table 35. Mapping table worksheet*

| From_input | To_name | (✓) DECODE | (✓) SEARCH | (✓) UPDATE | EXEC () | (✓) TEXT |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

## Filter Table Planning Worksheet

*Table 36. Filter table worksheet*

| DEFAULT=BLOCK\|PASS | | |
|---|---|---|
| **Alias name** | **Operator** | **Value** |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# 16 NetView AutoBridge Sample Members

This chapter lists and describes the sample members provided with AutoBridge. Use them to assist you in installing and customizing AutoBridge. These samples are shipped in the SEYLSPL library and are listed in the following sections:

- "Installation Samples" lists the AutoBridge installation samples. Use these samples when defining operators and their profiles, commands, and VSAM files.
- "Installation Verification Programs (IVPs)" on page 239 lists three IVPs for verifying your AutoBridge installation.
- "AutoBridge Table Samples" on page 239 lists the sample process, filter, and mapping tables, as well as sample initialization tables for both resident and remote NetViews.
- "User-written CLISTs and Panel Samples" on page 239 lists two samples of user-written CLISTs.
- "User-written Functions" on page 240 lists user-written functions that you can call from mapping table segments. These functions manipulate data extracted from a messages and MSUs.

## Installation Samples

**EYLCMD** The command model statements to add to the NetView DSICMD member. These statements should be added exactly as shown in the sample. This step is described in "Adding Command Model Statements to NetView" on page 148.

**EYLDMN** The command facility definitions to add to the NetView DSIDMN member. These statements should be added exactly as shown in the sample. This step is described in "Customizing the DSIPARM DSIDMN Member" on page 151.

**EYLPRFAO** The operator profile for the checkpoint tasks. This definition is an example only. If you change the profile name of the checkpoint task in the installation sample EYLRES, rename this member to reflect the new name. This step is described in "Creating Profiles for NetView Autotasks" on page 146.

**EYLPRFDH** The high-level operator profile for a NetView Bridge dispatcher. This definition is an example only. If you change the profile name of the NetView Bridge dispatcher in the EYLRES sample, rename this member to reflect that name. This step is described in "Creating Profiles for NetView Autotasks" on page 146.

**EYLPRFDL** The low-level operator profile for a NetView Bridge dispatcher. This definition is an example only. If you changed the profile name of the NetView Bridge dispatcher in the EYLRES sample, rename this member to reflect that name.

**EYLPRFRD** The operator profile for the remote NetView Bridge dispatcher. This definition is an example only. If you change the profile name of the remote dispatcher in either the EYLRES or EYLREM installation samples, rename EYLPRFRD to reflect the new name. This step is described in “Creating Profiles for NetView Autotasks” on page 146.

**EYLREM** The operator ID definitions for remote NetViews. Copy this member into the NetView DSIOPF member in each remote NetView. These definitions are examples only. Rename the autotasks as appropriate. Use as many autotasks as suits your needs.

**Note:** If you change the operator IDs, make a corresponding change in the RDISPATCH variable and BEGIN DISPATCHER variables of the initialization table (EYLATINT).

The sample contains definitions for one remote NetView Bridge dispatcher (required) and two checkpoint tasks. You may define a checkpoint task associated with any or all NetView Bridge dispatchers on the resident NetView.

**EYLRES** The operator ID definitions for the resident NetView. Copy EYLRES into the NetView DSIOPF member in the resident NetView. These definitions are examples only. Rename the autotasks as appropriate. Use as many autotasks as suits your needs.

**Note:** If you change the operator IDs, make a corresponding change in the RDISPATCH variable and BEGIN DISPATCHER variables of EYLATINT.

The sample contains definitions for one remote dispatcher (required if there are any remote NetViews) and for two NetView Bridge dispatchers and their two associated checkpoint tasks. You may define one to four NetView Bridge dispatchers, each with an associated checkpoint task. This step is described in “Adding Operator IDs for NetView Autotasks” on page 145.

**EYLSID01** The sample IDCAM to be used by EYLSJ008 to delete the current checkpoint file.

**EYLSI101** The sample IDCAM to be used by EYLSJ008 to create the new checkpoint file.

**EYLSJ002** The JCL job to allocate NetView Bridge adapter output data sets. Change these JCL statements as necessary to reflect the correct DASD type and data set names, as well as any other information that is unique to your environment. This step is described in “The User-Supplied Output Data Sets” on page 17.

**EYLSJ008** A JCL job that defines VSAM clusters for the AutoBridge checkpoint files. Change these JCL statements as necessary to reflect the correct DASD type and data set names, as well as any other information that is unique to your environment. This step is described in “Allocating the Checkpoint File VSAM Data Set” on page 150.

| | |
|---|---|
| **EYLATMEM** | Sample DSIDMN definition statements for initializing the checkpoint manager data services task. |
| **EYLATSUB** | Sample DSIDMN definition statements for initializing the table manager data services task. |

## Installation Verification Programs (IVPs)

After installing AutoBridge and customizing its tables, profiles, and other components, you can execute any of the following IVPs to produce simple BNJ146I messages. Copy these members into a concatenated DSICLD data set. Rename them if necessary.

| | |
|---|---|
| **EYLIVP1** | A sample IVP that produces a BNJ146I RECFMS message |
| **EYLIVP2** | A sample IVP that produces a BNJ146I generic alert message |
| **EYLIVP3** | A sample IVP that produces a BNJ146I non-generic alert message |

## AutoBridge Table Samples

**EYLATFIL** The sample filter table. You may customize the default value of PASS or BLOCK and the following filter conditions.

**EYLATINT** The sample initialization table for a resident NetView. You must customize this table to reflect the IDs and table names on your system.

**Note:** If you changed the operator IDs in the EYLRES sample, make a corresponding change in the RDISPATCH variable and BEGIN DISPATCHER variables of this table.

This step is described in "Creating Profiles for NetView Autotasks" on page 146.

**EYLATMAP** The sample mapping table. You can modify the mapping segments to suit your needs or use them as a model for creating others.

**EYLATPRO** The sample process table. You can modify the process segments to suit your needs or use them as a model for creating others.

**EYLINTRM** The sample initialization table for remote NetViews. You must customize this table to reflect the IDs and table names on your system.

**Note:** If you changed the operator IDs in the EYLREM sample, make a corresponding change in the RDISPATCH variable and BEGIN DISPATCHER variables of this table.

Rename this member to EYLATINT after copying it to a remote NetView.

## User-written CLISTs and Panel Samples

To use these samples, copy the EYLEXA00 and EYLEXUSR members into a concatenated DSICLD data set. Rename them if necessary. Copy the EYLKXUSR panel into a concatenated CNMPNL1 data set. If you rename EYLKXUSR, edit EYLEXUSR to reflect the new name on the VIEW command.

**EYLEXA00** A sample REXX exec that can be invoked in response to an unalert (an alert with description code point of A*nnn*). Customize the NetView Bridge dispatcher specified on the ABAPI invocation as necessary, as well as the description line to be updated in the record.

**EYLEXUSR** A sample REXX exec that allows user interaction with records on the NetView Bridge. This exec displays a panel (EYLKXUSR) showing the contents of some of the fields in a record. Edit the fields in this exec's CUSTOMIZE subroutine to use EYLEXUSR on your system. The remainder of the exec should work without modification.

**EYLKXUSR** A sample panel used with EYLEXUSR. You must customize the panel entries to agree with changes made in the CUSTOMIZE subroutine of the EYLEXUSR exec.

## User-written Functions

To use any of these functions, copy the members into a concatenated DSICLD data set. Rename them if necessary. If you change the name of a function, make a corresponding change to those statements in the sample mapping table that invoke that function.

**EYLEXAYR** This function adds the current year to a month/day string in the format *MM/DD*. The output string is in the format *MM/DD/YY*. Use this function when parsing a BNJ146I message such as:

```
MSGSTR(1,1) DATE_OCCURRED,EYLEXAYR(DATE_OCCURRED);
```

**EYLEXCDT** This function converts a hexadecimal date string in the format *YYMMDD* to a decimal string in the format *MM/DD/YY*. Use this function when parsing the date from MSU subvector 10 as in the following example:

```
MSUSEG(0000.01.10,3,3) DATE_OCCURRED,EYLEXCDT('DATE_OCCURRED'X);
```

**EYLEXCHG** This function returns a substitute text string for a string you specify. For example, you could substitute the string 'TOKEN RING TEMPORARY ERROR' with 'RECOVERED BEACONING CONDITION'. Use this exec when parsing the description from MSU subvector 92 as in the following example:

```
MSUSEG(0000.92,6,2) DESCRIPTION,DECODE,EYLEXCHG(DESCRIPTION);
```

**EYLEXCON** This sample function concatenates a text string to 45 characters for insertion into the DESCRIPTION field. Use it when you want to see at least part of the probable cause text in the Problem Reporter DESCRIPTION field. This could be used when parsing the probable cause from MSU subvector 93 as in the following example:

```
MSUSEG(0000.93,3,2) DESCRIPTION,DECODE,EYLEXCON(DESCRIPTION),UPDATE;
```

**EYLEXCTM** This function converts a hexadecimal time string in the format *HHMMSS* to a decimal string in the format *HH:MM*. Use this function when parsing the time from MSU subvector 10 as in the following example:

```
MSUSEG(0000.01.10,6,3) TIME_OCCURRED,EYLEXCTM('DATE_OCCURRED'X);
```

**EYLEXLAD** This function extracts the LAN Fault Domain Description MAC addresses and builds a Tivoli Information Management for z/OS list that is separated by commas. Use this function when parsing the LAN link connection from MSU subvector 51, subfield 06, to create a line/circuit numbers list in the symptom data:

```
MSUSEG(0000.51.06,3) S1422,EYLEXLAD('S1422'X);
```

**EYLEXLNM** This function extracts the LAN Fault Domain names and builds a Tivoli

Information Management for z/OS list that is separated by commas. Use this function when parsing the LAN link connection from MSU subvector 51, subfield 26, to create a device names list in the symptom data:

```
MSUSEG(0000.51.26,3) S1416,EYLEXLNM('S1416'X);
```

**EYLEXNAM** This function strips the device name returned from the HIER() function down to the 8-character name field (dropping the 4-character type). Use this function when parsing the HIER list in the MSU as in the following example:

```
HIER(4) DEVICE_NAME,EYLEXNAM(DEVICE_NAME),SEARCH;
```

**EYLEXSTR** This function informs AutoBridge that the input is in hexadecimal format. AutoBridge's PARSE support converts such data into character format so that it is acceptable to Tivoli Information Management for z/OS. Use this function when parsing the LAN link connection from subvector 51, subfield 07, to save the beaconing type in the symptom data abstract:

```
MSUSEG(0000.51.07,3) S0C3A,EYLEXSTR('S0C3A'X);
```

17

# Tivoli Information Management for z/OS to Tivoli NetView Connection

Tivoli Information Management for z/OS extends problem control and management to remote parts of your network. Because Tivoli Information Management for z/OS NetView AutoBridge provides an interface to Tivoli NetView for z/OS, you can automatically open and update network problems in Tivoli Information Management for z/OS's database. You can also connect Tivoli Information Management for z/OS with Tivoli NetView running on an AIX® platform through the Tivoli NetView for z/OS program. This connection creates centralized network management within a distributed, multivendor, heterogeneous environment, as shown in Figure 14.

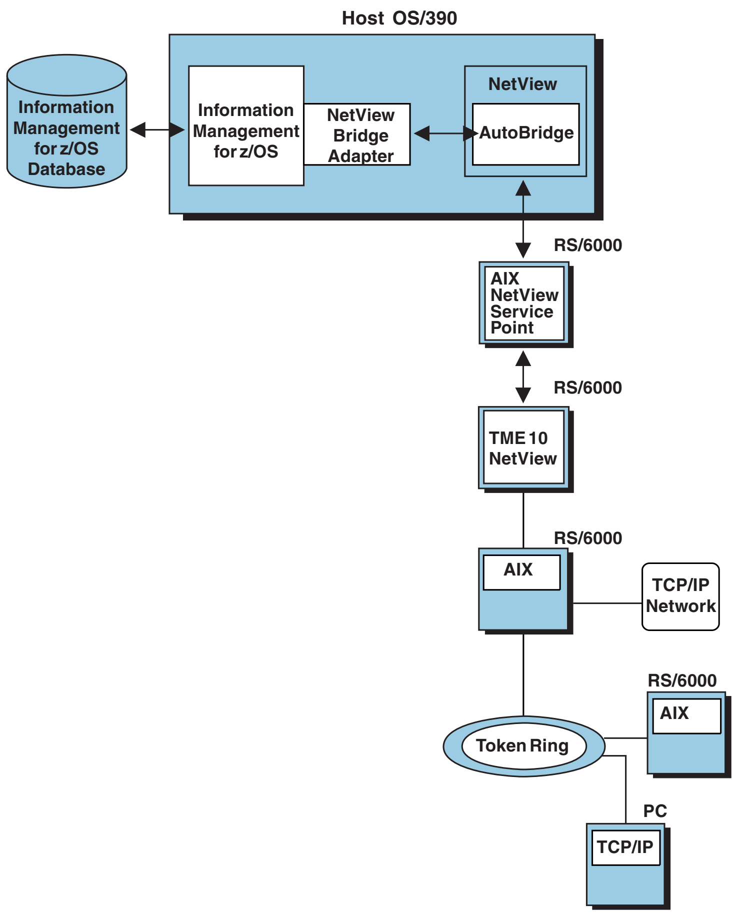


*Figure 14. Centralized Network Management within a Heterogeneous Environment*

## Understanding the Tivoli NetView Connection

The Tivoli NetView for z/OS program connects to Tivoli NetView (previously known as NetView for AIX) using the AIX NetView Service Point program. The Tivoli NetView program filters the Simple Network Management Protocol (SNMP) traps received from TCP/IP networks, converts them to SNA alerts, and sends them to Tivoli NetView for z/OS. The Tivoli NetView for z/OS program uses Tivoli Information Management for z/OS NetView AutoBridge which in turn uses the Tivoli Information Management for z/OS NetView Bridge Adapter to send requests to Tivoli Information Management for z/OS.

**Note:** The connection between Tivoli NetView for z/OS and Tivoli NetView is available only from the Tivoli NetView program that runs on the AIX operating system.

### What is Tivoli NetView for AIX?

The Tivoli NetView program is a comprehensive management tool for heterogeneous, multivendor devices on TCP/IP networks.

The Tivoli NetView program provides configuration, fault, security, and performance management functions, along with many features that make it easy to install and use. It provides an open network management platform that enables the integration of Simple Network Management Protocol (SNMP) and Common Management Information Protocol (CMIP) applications. For cooperative management of TCP/IP networks, Tivoli NetView uses the AIX NetView Service Point program to communicate with your Tivoli NetView program. The Tivoli NetView program is a network and system management tool that provides distributed or centralized management for your network.

### What is AIX NetView Service Point?

The AIX NetView Service Point program is a network management tool that enables you to expand the Tivoli NetView for z/OS centralized management of an SNA network to include non-SNA network devices, by providing a gateway to Tivoli NetView on an RS/6000®. Tivoli NetView for AIX is an example of a program that uses AIX NetView Service Point to enable Tivoli NetView for z/OS to communicate with non-SNA devices.

AIX NetView Service Point and Tivoli NetView for z/OS are separate programs. Tivoli NetView for z/OS is a host program that provides network management functions in an SNA network. AIX NetView Service Point is a library of functions and a set of system services that enable applications residing on a local or distributed AIX-based workstation to exchange data with Tivoli NetView for z/OS.

When you install and customize AIX NetView Service Point, the product functions as a gateway. The Service Point does not provide a local operator interface. AIX NetView Service Point runs in an unattended environment.

The AIX NetView Service Point program operates in the AIX and UNIX environments and takes advantage of the multi-user, multiprocessing, and network facilities provided by AIX and UNIX.

AIX NetView Service Point enables the host program and the Tivoli NetView program to exchange SNA management services (MS) major vectors over SNA Services/6000 or SNA Server/6000. Tivoli NetView supplies the spappld daemon as a service point application and starts the daemon as a part of the Tivoli NetView initialization process. The resulting connection enables a host operator to send SNA MS Execute (X'8061') Major Vectors containing RUNCMD commands. Tivoli NetView processes the contents of the RUNCMD command in the SNMP environment.

For information about configuring the AIX NetView Service Point, refer to the *AIX NetView Service Point Installation, Operation, and Programming Guide*.

## Software Requirements

Using the host connection between Tivoli NetView and Tivoli NetView for z/OS requires the following software programs.

- One of the following host z/OS programs:
  - Tivoli NetView for z/OS Version 1 Release 1.
  - NetView Version 1 Release 3, Version 2 Release 3, or Version 2 Release 4.
  - NetView for MVS Version 3 with the Enterprise feature.
- AIX NetView/6000 Version 2 Release 1, NetView for AIX Version 3 or 4, or Tivoli NetView Version 5.
- One of the following AIX NetView Service Point programs:
  - AIX NetView Service Point Version 1 Release 1 or Version 1 Release 2 Modification 0 to use with SNA Services/6000
  - AIX NetView Service Point Version 1 Release 2 Modification 1 or later, to use with SNA Server/6000
- AIX SNA Services/6000 Version 1 Release 2, or AIX SNA Server/6000 Version 2 Release 1 or later

## Purpose of the Host Connection

The purpose of the connection between Tivoli NetView and Tivoli NetView for z/OS is to inform the host program of certain events in a TCP/IP network by converting selected traps into alerts and forwarding them to the host program. The host program can respond to the alert by returning a RUNCMD command that contains an appropriate response to the event.

The host connection also enables Tivoli NetView for z/OS to issue a command to be run in the SNMP environment. The command is enclosed in a RUNCMD command and sent to the SNMP environment for processing. The results are returned to the host program in another RUNCMD command.

## How the Host Connection Works

You can use the AIX NetView Service Point program with the Tivoli NetView and Tivoli NetView for z/OS programs to cooperatively manage both SNA networks and TCP/IP networks. The AIX NetView Service Point program acts as a bridge between the Tivoli NetView program and the host enabling the Tivoli NetView for z/OS program to use the facilities of the Tivoli NetView program to manage SNMP devices.

The Tivoli NetView tralertd daemon receives the traps that meet the filtering criteria. ***Traps*** are a type of event in which agents send information to the manager without an explicit request from the manager. Tivoli NetView uses the NetView Service Point application programming interfaces (APIs) to convert the traps into alerts or network management vector transports (NMVTs). The NetView Service Point program then forwards the alerts to Tivoli NetView for z/OS. From Tivoli NetView for z/OS, the Tivoli Information

Management for z/OS NetView AutoBridge program processes the NMVTs and uses the data from the NMVTs to create and update the Tivoli Information Management for z/OS records.

The host program responds to the alert by enclosing an appropriate command in a RUNCMD command. The RUNCMD command is started by the Tivoli NetView program, which then returns a response to the host program.

The host connection also enables you to use Tivoli NetView for z/OS automation facilities to invoke shell scripts or send SNMP network management commands with RUNCMDs through the AIX NetView Service Point program to the Tivoli NetView program to monitor and manage your internet environment.

For more detailed information about how to implement and maintain the connection between Tivoli NetView and Tivoli NetView for z/OS, refer to *Tivoli NetView Host Connection*.

## Advantages of Connecting Tivoli Information Management for z/OS to Tivoli NetView

Connecting Tivoli Information Management for z/OS to Tivoli NetView enables you to get an enterprise view of your network. You can create and update problem records in Tivoli Information Management for z/OS databases in remote locations and dynamically track problems in your distributed network. This connection also enables you to create distributed and host oriented problem records in one database and automate tasks to ensure accuracy and improve the quality of your management system. This allows time for trend analysis and more effective network planning.

# III — Problem Service

18

# Understanding the Problem Service Component

The Problem Service component provides distributed help desk applications with an interface to the Tivoli Information Management for z/OS database on MVS so that both products can be part of the same distributed networking environment. Tivoli Information Management for z/OS enables applications on remote environments to connect to the Tivoli Information Management for z/OS system on MVS and to access the Tivoli Information Management for z/OS database.

Problem Service consists of the following components:

- Tivoli-related functions that install the Problem Service object and operations into the Tivoli database
- Daemon-related functions: the executable Problem Service daemon and user exits
- Sample files that provide coding examples for the Problem Service operations, configuration file, MVS tables, and high-level application program interface (HLAPI) profile
- Files needed to compile and link-edit an application that will use Problem Service operations

Conceptually, Problem Service provides:

- Session management with Tivoli Information Management for z/OS
- A set of application programming interfaces (APIs) that enable you to interact with a Tivoli Information Management for z/OS database
- A data mapping facility that enables you to define data manipulations

## Problem Service Sessions

The sessions provided by Problem Service are determined by what you specify in the Problem Service configuration file. These sessions provide an application access through the HLAPI to a Tivoli Information Management for z/OS database. There are three types of sessions available:
- Outbound
- Reverse
- Monitor

Multiple outbound sessions can be specified in the configuration file, but only one reverse session and one monitor session are permitted. Figure 15 on page 252 shows an overview of

Problem Service sessions, including an interface with an application and a Tivoli Information Management for z/OS database.

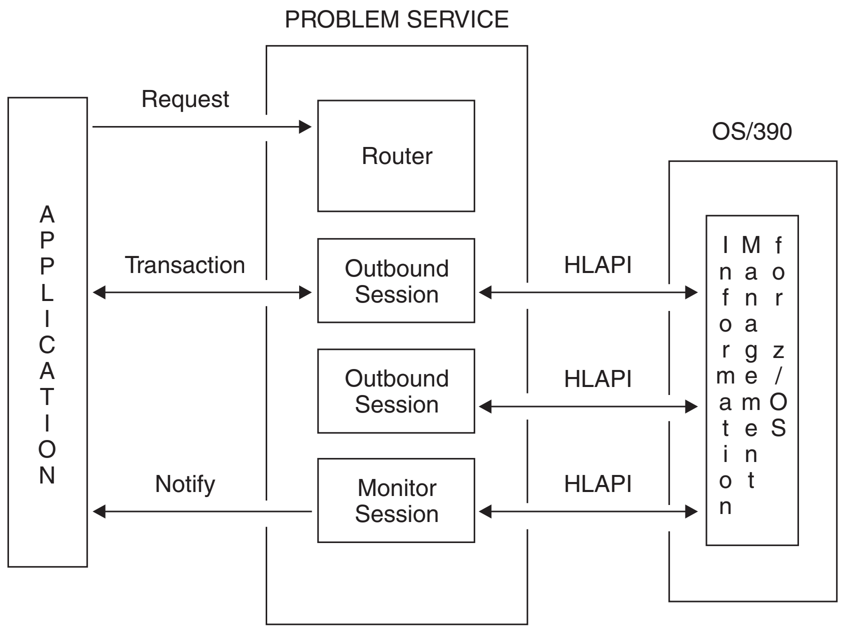


*Figure 15. Problem Service Sessions Example*

With this example, only three sessions were specified in the configuration file: one monitor session and two outbound sessions. A reverse session was not defined.

When the monitor session (polling daemon) detects a change in a database record that was assigned to or created by the application, it notifies the application.

The two outbound sessions are used for Problem Service operations initiated by the application. When the application requests an operation, Problem Service routes the request to the first available outbound session. A transaction interface is established and Problem Service performs the operation requested.

## Problem Service Operations

Problem Service enables you to use your applications to share information in Tivoli Information Management for z/OS database records. The following operations, using outbound sessions, enable you to exchange information between your applications and Tivoli Information Management for z/OS databases:

- Checkin
- Checkout
- Delete
- Propagate
- Retrieve
- Search
- Transfer
- Update

The following are Problem Service control operations:

- Ping

- Shutdown

The following are Problem Service automated operations:
- Reverse assignment
- Monitor

The operations provided by Problem Service are affected by the availability of the network connection between the Problem Service daemons and Tivoli Information Management for z/OS. If the network connection is unavailable, you cannot perform or complete some Problem Service operations. You can resume them when the connection becomes available.

Furthermore, the data fields handled by your application can differ in meaning and format from the data fields in Tivoli Information Management for z/OS records. To ensure data integrity when these records are interchanged between your application and Tivoli Information Management for z/OS, Problem Service provides customizable data mappings to map your application record fields to Tivoli Information Management for z/OS record fields and vice versa.

To maintain data integrity, avoid multiple update access to the same record. Use the Problem Service record locking capability to ensure that your application keeps others from updating a record that you want to update.

A description of each Problem Service operation follows. Each description contains a prototype statement for the operation in the form of:

```
  Response   Operation   Input
```

where Input consists of

```
 (in datatype datavalue)
```

For example code fragments that show how to use the operations, see "Coding Examples for Problem Service Operations" on page 300.

## Unlocking a Record

The checkin operation unlocks a specified record in the Tivoli Information Management for z/OS database by checking it in. The prototype statement for checkin is:

```
void checkin (in string rnid);
```

**Where**:

**void** Indicates that there are no returned values.

*rnid* Is a 1 to 8-character string that is the record number of the record to be checked in to a Tivoli Information Management for z/OS.

## Locking a Record

The checkout operation locks a specified record in the Tivoli Information Management for z/OS database by checking it out. The prototype statement for checkout is:

```
void checkout (in string rnid);
```

**Where**:

**void** Indicates that there are no returned values.

*rnid* Is a 1 to 8-character string that is the record number of the record to be checked out of Tivoli Information Management for z/OS.

## Deleting Records

The delete operation deletes a specified record in Tivoli Information Management for z/OS. The prototype statement for delete is:

```
void delete (in string rnid);
```

**Where**:

**void** Indicates that there are no returned values.

*rnid* Is a 1 to 8-character string that is the record number of the record to be deleted from Tivoli Information Management for z/OS.

## Propagating Records

The propagate operation creates a copy of the record in the Tivoli Information Management for z/OS database. When created in checked-out status, a Tivoli Information Management for z/OS user on the host can only view the record to see any updates made by a user from the workstation.

You can make this record a read-only copy by supplying the checkout indication data field or hardcoding the checkout indicator in the Problem Service configuration file data mappings for propagate.

The prototype statement for propagate is:

```
string propagate (in GWAttrList keyvaluepairs, in string recordtypevalue);
```

**Where**:

*string* Is the Tivoli Information Management for z/OS record number returned when the operation has successfully completed.

*keyvaluepairs*
Is a sequence of name value pairs consisting of name and data values.

*recordtypevalue*
Is the name of the record type that corresponds to the configuration file mapping section identified by the RecordTypeValue statement.

The record is assumed to exist if a record number (*rnid*) is passed as one of the name value pairs. The field that contains the *rnid* is determined by the ForeignIMRNIDField configuration file statement. In this case, the propagate operation attempts to update the existing Tivoli Information Management for z/OS record.

Your application can pass only changed data or it can pass all record data. The InputJustChangedData statement in the Problem Service configuration file defines the selected behavior to Problem Service.

When you pass all data and use data mapping, Problem Service attempts to delete all NULL data fields from the Tivoli Information Management for z/OS record. When you pass only changed values and you want to delete a Tivoli Information Management for z/OS field, pass NULL as the data.

## Retrieving Records

The retrieve operation retrieves a specific Tivoli Information Management for z/OS record. The prototype statement for retrieve is:

```
GWAttrList retrieve (in string rnid, in string recordtypevalue);
```

**Where**:

**GWAttrList**
: Is a Tivoli Information Management for z/OS record in a sequence of name value pairs returned in response to the retrieve operation.

*rnid*
: Is a 1 to 8-character string that is the record number of the record to be retrieved from Tivoli Information Management for z/OS.

*recordtypevalue*
: Is the name of the record type that corresponds to the configuration file mapping section identified by the RecordTypeValue statement.

## Searching for a Record

The search operation returns a list of Tivoli Information Management for z/OS records that match the specified search criteria. The prototype statement for search is:

```
SearchResultList search (in GWAttrList keyvaluepairs, in string recordtypevalue);
```

**Where**:

**SearchResultList**
: Is a sequence of elements that match the search criteria of the search operation. Each element contains the *rnid* and a data value, which is typically the data abstract, for each Tivoli Information Management for z/OS record. The AssociatedDataField statement in the Problem Service configuration file determines the Tivoli Information Management for z/OS data field that is returned in the name value pair. The number of matches is limited to the value of the SearchHits statement in the Problem Service configuration file.

*keyvaluepairs*
: Is the sequence of name value pairs consisting of name and data values which specify the criteria to be used for the search operation.

*recordtypevalue*
: Is the name of the record type that corresponds to the configuration file mapping section identified with the RecordTypeValue statement.

## Transferring Records

The transfer operation creates a record in the Tivoli Information Management for z/OS database that can be viewed and updated by a Tivoli Information Management for z/OS user on the host. The prototype statement for transfer is:

```
string transfer (in GWAttrList keyvaluepairs, in string recordtypevalue);
```

**Where**:

*string*
: Is the Tivoli Information Management for z/OS record number returned when the operation has successfully completed.

*keyvaluepairs*
: Is a sequence of name value pairs consisting of name and data values. The *rnid* can be specified as one of the name/value pairs or it can be assigned by Tivoli Information Management for z/OS. When Tivoli Information Management for z/OS assigns the *rnid*, it is always an 8-character numeric string.

*recordtypevalue*
: Is the name of the record type that corresponds to the configuration file mapping section identified by the RecordTypeValue statement.

## Updating Records

The update operation updates the specified Tivoli Information Management for z/OS record. The prototype statement for update is:

```
void update (in GWAttrList keyvaluepairs, in string recordtypevalue);
```

**Where**:

**void** Indicates there are no returned values.

*keyvaluepairs*
Is the sequence of name value pairs that consist of name and data values. One of the name value pairs must contain an *rnid*. The field that contains the *rnid* is determined by the ForeignIMRNIDField configuration file statement.

*recordtypevalue*
Is the name of the record type that corresponds to the configuration file mapping section identified by the RecordTypeValue statement.

Your application can pass only changed data or it can pass all record data. The InputJustChangedData statement in the Problem Service configuration file defines the selected behavior to Problem Service.

If you pass all data and use data mapping, Problem Service attempts to delete all NULL data fields from the Tivoli Information Management for z/OS record. When you pass only changed values and you want to delete a Tivoli Information Management for z/OS field, pass NULL as the data.

## Control Operations

Here are the descriptions for the control operations (ping and shutdown) provided by Problem Service.

### Pinging for Status

Ping is used by your application to obtain the status of the Problem Service daemons. The prototype statement for ping is:

```
PingResult ping ();
```

The operation returns *PingResult*, which indicates the status of Problem Service outbound, reverse assignment, and monitor sessions. If Problem Service is not running, an exception is thrown.

### Shutting Down Problem Service

Shutdown closes the Problem Service and its associated child daemons. This is a hard shutdown, and does not ensure that all transactions are completed. The prototype statement for shutdown is:

```
void shutdown ();
```

No values are returned.

## Automated Operations

The remaining operations provided by Problem Service are automated operations. You tailor the Problem Service configuration file to customize the behavior of the operations. There can only be one session specified for each automated operation in the configuration file: one reverse session and one monitor session.

## Reverse Assignment Operation

Reverse assignment allows records created in Tivoli Information Management for z/OS to be used and assigned in the local database. You might want to transfer the responsibility of specific Tivoli Information Management for z/OS records to your distributed help desk staff or have your staff share the responsibility for those records with the Tivoli Information Management for z/OS staff on the host. Periodically, reverse assignment automatically queries the Tivoli Information Management for z/OS database and identifies records that meet specific search criteria.

The record is still owned by Tivoli Information Management for z/OS and the record can be updated by the Tivoli Information Management for z/OS staff on the host. When you update a reverse assigned record, the record should be checked-out and retrieved from the Tivoli Information Management for z/OS database before it is updated.

Reverse assigned records can be monitored for changes. See “Monitor Operation” on page 258 for more information on monitoring.

### Tailoring

You tailor the Problem Service configuration file to customize the behavior of this operation by specifying:

- The time interval between each activation of the reverse assignment operation.
- The search criteria to use when querying the Tivoli Information Management for z/OS database.
- The executable to be invoked when search matches are found.

### Activating

The reverse assignment automated operation is activated at the end of each time interval. If the network connection to Tivoli Information Management for z/OS is not available at activation time, the reverse assignment operation is not performed for that activation.

If the network connection is available at activation time, reverse assignment queries the Tivoli Information Management for z/OS database according to the specified search criteria. When it queries the database, it considers all records except those that have already been transferred or reverse assigned by your application’s system. The number of search matches is limited to the SearchHits configuration file value.

### Record Processing

For each record that is identified in the Tivoli Information Management for z/OS database, the executable identified by the RACallApp statement is run on the system identified by the ForeignHost statement. These statements and their values can be found in your configuration file.

The record number, record type, and the reverse assignment GatewayID (R*GatewayID*) are passed to the executable, which performs the processing of the Tivoli Information Management for z/OS record. This processing could include:

1. Checking out the record.
2. Retrieving the record.
3. Performing application-specific actions (for example, creating a record in the distributed help desk’s database).

4. Updating the Tivoli Information Management for z/OS record to add the gateway ID to mark the record as being reverse assigned by this Problem Service. This enables the monitor operation on the record.

   **Note:** This step prevents the reverse assignment operation from repeatedly processing the same record.

5. Checking in the record.

## Monitor Operation

Problem Service provides an automated monitor operation that periodically queries the Tivoli Information Management for z/OS database to identify records that have been transferred or reverse assigned by your distributed help desk and that have been recently updated in Tivoli Information Management for z/OS.

### Tailoring

You tailor the Problem Service configuration file to specify:

- The time interval between each activation of the monitor operation.
- The executable to be invoked when search hits are found.

### Activating

The monitor operation is activated at the end of each time interval. If the network connection to Tivoli Information Management for z/OS is not available at activation time, the monitor operation is not performed for that activation.

The monitor operation affects only the records that have been identified as being reverse assigned or transferred by your application's system. The records processed by Problem Service contain a gateway identifier, specified in the Problem Service configuration file, that associates them with your application's system that is running Problem Service.

**Note:** This assumes that the transfer data mappings include the Problem Service identifier and that you updated reverse assigned records with the Problem Service identifier.

The monitor operation selects only records that were last changed by a different user. All records last changed by your application through a Problem Service instance will be marked as being changed last by the gateway identifier value.

You must update the Tivoli Information Management for z/OS record or the monitor will always find the same record on the next monitor iteration. If you do not have data to change, you can use a field returned by the retrieve operation as the data to use for the update.

### Record Processing

For each record that the monitor finds in the Tivoli Information Management for z/OS database that matches the search criteria, the executable identified by the MonCallApp statement is run on the system identified by the ForeignHost statement. These statements and their values are in your configuration file.

The record number and record type are passed to the executable. The number of search matches is limited to the SearchHits configuration file value. The executable processes the Tivoli Information Management for z/OS record, which could include:

1. Checking out the record.
2. Retrieving the record.
3. Performing an application-specific function (for example, updating a record in the distributed help desk's database).
4. Updating the Tivoli Information Management for z/OS record to mark the record as being last altered by this instance of Problem Service.

   **Note:** This step prevents the monitor operation from repeatedly processing the same record.
5. Checking in the record.

## Problem Service Data Mappings

When a propagate or transfer operation copies a record to Tivoli Information Management for z/OS, it uses the data you pass to it as input for the Tivoli Information Management for z/OS record fields. Correspondingly, when you retrieve a Tivoli Information Management for z/OS record, it uses the data in the Tivoli Information Management for z/OS record fields as input. However, the record fields in your application's data might differ in number, name, meaning, and format from the Tivoli Information Management for z/OS record fields. You must define how these record contents are mapped from one database to the other.

The Problem Service data mappings consist of mapping rules, specified in the Problem Service configuration file, that the Problem Service operations apply when manipulating records. You can customize the data mappings for each operation to specify:

- Which of your data fields and Tivoli Information Management for z/OS record fields are processed for various operations.
- The association between your data fields and Tivoli Information Management for z/OS record fields that have similar meaning.
- The conversion mechanism to use to transform the field data from your format to the Tivoli Information Management for z/OS record format and from the Tivoli Information Management for z/OS record format to your application's data format.

You can disable all data mapping by specifying `PerformDataMapping=no` in the configuration file. When data mapping is disabled, you work with only Tivoli Information Management for z/OS field names and Tivoli Information Management for z/OS format data.

# 19 Problem Service Installation

This chapter explains how to plan for and install Problem Service.

Follow these steps to install and configure Problem Service:

1. Ensure that the prerequisite software is installed, configured, and operational on your workstation and on the MVS host.

   **Note:** With this release of Tivoli Information Management for z/OS, support for the use of Problem Service on a Sun Solaris platform is not included.
2. Install Problem Service on your workstation.

   **Note:** The Problem Service daemon (AIX) or the Windows NT® service must be installed on a Tivoli Management Region (TMR) server.
3. Configure Problem Service on your workstation (see "Customizing Your Problem Service Configuration File" on page 271).

   **Note:** Work with the Tivoli Information Management for z/OS system administrator to complete the Problem Service configuration.

## Planning for Problem Service Installation

The following sections describe how Problem Service fits into the Tivoli Information Management for z/OS environment and provides you with an overview of the Tivoli Information Management for z/OS setup that is necessary for operating Problem Service.

### Tivoli Information Management for z/OS Environment

To perform its operations, Problem Service uses the Tivoli Information Management for z/OS HLAPI to remotely access the Tivoli Information Management for z/OS database records from the workstation environment.

The Tivoli Information Management for z/OS HLAPI is a transaction-based application programming interface (API). It enables applications in remote environments to establish HLAPI working sessions with the host Tivoli Information Management for z/OS system to manipulate database records.

For more information on the Tivoli Information Management for z/OS HLAPI, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

### HLAPI Client

In the workstation environment, Tivoli Information Management for z/OS provides a HLAPI client that establishes communication links with the Tivoli Information Management for z/OS system on MVS. This feature of Tivoli Information Management for z/OS is made up of two interfaces: a requester and a client. Both interfaces must be installed and configured in the workstation environment.

#### Requester Interface

The requester interface provides the communication link to the Tivoli Information Management for z/OS system on MVS. Either TCP/IP or advanced program-to-program communication (APPC) is used, depending on the platform used.

#### Client Interface

The client interface must reside on the Problem Service workstation. It provides Problem Service with the interface to the Tivoli Information Management for z/OS HLAPI.

For more information on setting up the HLAPI client/server environment, refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide*.

## Installation Requirements

To install and configure Problem Service on your workstation, you need the hardware, storage, and software listed in the following sections.

### Hardware Requirements

The Problem Service option has no unique hardware requirements.

### Disk Space Requirements

In addition to your workstation's other requirements, the free disk space required for Problem Service is shown in Table 37.

*Table 37. Free Disk Space Requirements*

| Problem Service | Disk Space |
|---|---|
| AIX | 12 MB |
| Windows NT | 7 MB |

### Software Requirements

The installation and operation of Problem Service requires software on both MVS and the workstation.

#### MVS Host

To use Problem Service, you must have Tivoli Information Management for z/OS Version 7.1 installed and operational on an MVS host. Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for more information.

The Tivoli Information Management for z/OS HLAPI and either a remote environment server (RES) or multiclient remote environment server (MRES) must be set up on the Tivoli Information Management for z/OS system that owns the database that your Problem Service must access.

Both the *Tivoli Information Management for z/OS Client Installation and User's Guide* and the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* contain additional information on the Tivoli Information Management for z/OS environment setup for remote HLAPI applications.

### AIX Workstation

The following software is required for the installation, operation, and maintenance of Problem Service:

- AIX 4.2
- Tivoli Information Management for z/OS HLAPI Client for AIX
- Tivoli Management Environment 3.1 or higher

  **Note:** When writing an application to use Problem Service, you also need the Tivoli Application Development Environment.

For instructions on how to install and configure the HLAPI Client for AIX, refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide*. Install and configure both the requester and the client interface components on the AIX workstation.

**Note:** The client interface must reside on the same workstation where Problem Service is to be installed.

After you install and configure the HLAPI Client for AIX, record the name and location of the database profile you configure. You need this information for configuring Problem Service. If you did not change the name and location of this file, you can find it in the **/usr/lpp/idbhlapi/examples** directory. The default name is **idbdb.pro**. You can also customize and use the example HLAPI profile **gateway.prf** supplied with Problem Service.

### Windows NT Workstation

The following software is required for the installation, operation, and maintenance of Problem Service:

- Windows NT 4.0
- Tivoli Information Management for z/OS HLAPI for Windows NT Client feature
- Tivoli Management Environment 3.1 or higher

  **Note:** When writing an application to use Problem Service, you also need Tivoli Application Development Environment.

For instructions on how to install and configure the HLAPI for Windows NT Client, refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide*. Install and configure both the requester and the client interface components on the Windows NT workstation where Problem Service is to be installed.

After you install and configure the HLAPI for Windows NT Client, record the name and location of the database profile you configure. You need this information for configuring Problem Service. If you did not change the name and location of this file, you can find it in the **c:\infoapi\sample** directory. The default name is **database.pro**. You can also customize and use the example HLAPI profile **gateway.prf** supplied with Problem Service.

## Installing Problem Service

Before installing Problem Service, ensure that the HLAPI client can communicate with Tivoli Information Management for z/OS on MVS.

Some of the commands that you are requested to use in this section require you to be in the Tivoli environment. For AIX, entering the Tivoli environment is typically done by invoking the **setup_env.sh** shell script in the session being used. For Windows NT, entering the Tivoli environment is typically done by running the **setup_env.cmd** command file.

Problem Service is shipped on the Tivoli Information Management for z/OS installation CD-ROM in the **\Tivoli_Int\PS** directory. Insert the CD-ROM into your CD-ROM drive. Then use the Tivoli desktop to install Problem Service on your TMR server as follows:

1. On the Tivoli desktop, select **Desktop → Install → Install Product.** The Install Product window will appear.
2. On the Install Product window, select **Select Media**. The File Browser window will appear.
3. On the File Browser window, in the Path Name field, type the path of the Problem Service directory on the Tivoli Information Management for z/OS installation CD-ROM. For Windows NT, this is **x:\Tivoli_Int\PS** where **x:** is the CD-ROM drive letter; for AIX, this is **/cdrom/Tivoli_Int/PS**. Then select **Set Media and Close**.
4. Back on the Install Product window, select **TSD390 1.2 — Problem Service** as the product to be installed. When the Install Options window appears, you can specify the directory where the Problem Service sample programs will be stored. Select **Set** to close the Install Options window.
5. On the Product Install window, select the client on which you wish to install. The select **Install & Close**.
6. On the Product Install Product window, select **Continue Install**. When the installation is complete, a message will be displayed in the Product Install window. The files used by Problem Service are installed in the **$INST_DIR/$INTERP/InfoMgt/InfoGateway** directory.

To ensure that the installation completed successfully, enter the following command from the Tivoli environment to obtain the object ID (OID) of the object:

```
wlookup -r InfoMgtGW -a
```

The Info_GW instance is displayed, labeled with the OID. Your application uses the OID of the Info_GW instance to invoke Problem Service operations.

## National Language Support (NLS) for Messages

After installing Problem Service, the message catalog resides in the **/msg_cat/C** directory under the Tivoli install tree. For AIX, this directory needs to be part of your NLSPATH.

### AIX Workstations

For catalog files to reside in your NLSPATH, issue the **export LANG=C** command after you have entered the Tivoli environment. When the **bl*.cat** files reside in the NLSPATH, the messages issued by the daemon resolve correctly.

### Windows NT Workstations

LOCPATH points to the directories used to convert data between different code sets. For example, you can set the LOCPATH variable by specifying:

```
LOCPATH=C:\INFOAPI\LOCALE
```

The LOCPATH variable must be set in the system environment variables on the TMR server where Problem Service is installed. You must be logged on as an administrator to update system environment variables.

## REGSRV2 Program (Windows NT Only)

Problem Service is automatically registered as a Windows NT service as part of the installation process. If it should fail for any reason, or if you need to register or unregister Problem Service, information on running REGSRV2 is provided here. You can change the drive letter and directory path if necessary.

To install Problem Service (gw_nxd) as a Windows NT service, use the REGSRV2 program. The format is:

```
REGSRV2 x app_name
```

**Where:**

*x* Is I to install the service or D to delete the service.

*app_name*
Is the full application path name of the program, when the Windows NT service is being installed. If the full application path name contains a blank, enclose it in quotes. For example:

```
REGSRV2 I "C:\Program Files\GW_NXD.EXE"
```

The *app_name* parameter is not needed when deleting the Windows NT service. For example:

```
REGSRV2 D
```

**Note:** You must be logged on as an administrator to install the Windows NT service.

20

# Planning for Problem Service Configuration

This chapter provides a general overview of some basic Tivoli Information Management for z/OS concepts and the process for configuring Problem Service.

## Basic HLAPI Concepts

The Problem Service configuration requires you to be familiar with the following HLAPI-related concepts:

- HLAPI transactions
- The input and control Parameter Data Block (PDB), used for HLAPI transactions
- HLAPI data views that consist of either:
  - Program interface data tables (PIDTs) and program interface pattern tables (PIPTs)
  - Data model records
- Program interface alias table (PALT)

This section provides introductory information about these concepts. For a more detailed description, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

### HLAPI Transactions

HLAPI is a transaction-based application program interface. Problem Service uses HLAPI transactions to establish sessions with the Tivoli Information Management for z/OS system and to manipulate its database records.

In particular, to perform the Problem Service operations as described in “Problem Service Operations” on page 252, Problem Service uses the HLAPI transactions:
- checkin
- checkout
- create
- inquiry
- retrieve
- update

For example, the Problem Service propagate and transfer operations use the HLAPI create transaction to create a corresponding record in the Tivoli Information Management for z/OS database. The propagate operation also uses an implicit checkout on the create transaction to lock the record in the Tivoli Information Management for z/OS database. This protects the propagated record from being updated by the Tivoli Information Management for z/OS staff or other users and allows only the application that propagated the record to update it.

## HLAPI PDBs

The HLAPI PDBs contain parameters that control the operating characteristics of the HLAPI sessions that Problem Service uses to connect to the Tivoli Information Management for z/OS system and to manipulate the records in the database. Problem Service uses these PDBs for the HLAPI transactions.

## HLAPI Data Views

HLAPI uses data views to define the Tivoli Information Management for z/OS record fields that an application can access using the HLAPI transactions. These views can be PIDTs and PIPTs or data model (data view, data attribute, and data validation) records.

In a Tivoli Information Management for z/OS database that is not customized, PIDTs and associated PIPTs define the base Tivoli Information Management for z/OS records. You can customize these PIDTs and PIPTs to define additional user-defined fields in the Tivoli Information Management for z/OS records that you want your application to manipulate. You could also build data model records that define your fields.

Each HLAPI transaction has a separate PIDT that is also associated with a specific Tivoli Information Management for z/OS record type. For example, the HLAPI create transaction has a separate PIDT that defines the fields that can be used when creating a record in the Tivoli Information Management for z/OS database.

If you use data model records, the data view record can apply to multiple record transactions. For example, you can use the same data view for HLAPI create, update, or retrieve transactions.

### Structured and Prefix Word Indexes

The PIDT or data attributes identify specific fields within a Tivoli Information Management for z/OS record using Tivoli Information Management for z/OS ***structured word*** (s-word) and ***prefix word*** (p-word) indexes. An s-word index is represented by an S, followed by 4 hexadecimal characters. An example of an s-word index is `S0B5C`. A p-word index is represented by a P, followed by 4 hexadecimal characters. An example of a p-word index is `P028A`.

### P-Words

You should be familiar with p-words and their uses. A p-word consists of a keyword that performs searches on fields in Tivoli Information Management for z/OS database records. It can be associated to one or several s-word indexes. A p-word can be up to 6 characters long and must include the slash (/) or underscore (_) character as the last character. An example of a p-word is `AUTH/`.

To specify Tivoli Information Management for z/OS fields that HLAPI recognizes in the Problem Service configuration file, you can use indexes, alias names, and in some cases, p-words. For simplicity, the sample Problem Service configuration file uses s-word indexes instead of alias names.

## HLAPI PALTs

The PALT enables applications to specify alias names for PIDTs, p-words, p-word indexes, and s-word indexes. In a PALT, you can specify default values for the Tivoli Information

Management for z/OS record fields. That is, you can specify default response data values for Tivoli Information Management for z/OS record fields for which your application does not provide a response value.

You can define an alias name for a field, and use the alias name instead of using an s-word or a p-word index to identify the field. An alias name can as long as 32 characters. For example, you could define and use an alias name of `STATUS` instead of using the s-word index of `S0BEE`.

## Problem Service Configuration Process

Problem Service configuration is a multistep process that must be done to enable Problem Service operations. This process may involve customizing the Tivoli Information Management for z/OS system.

**Note:** Work with the Tivoli Information Management for z/OS administrator to complete the Problem Service configuration process.

### Sample Configuration File

A configuration file used by Problem Service controls various aspects of how Problem Service operates in your application's system and environment. It is a root-owned file and access to it is usually restricted to your application's administrator.

The sample configuration file **blmygc.cfg** is provided with Problem Service, but it must be customized before use. The **blmygc.cfg** file is made up of a series of statements that are grouped into three main parts:

1. Tivoli Information Management for z/OS HLAPI session information
2. Problem Service general settings
3. Data mappings

This file is located in the **$INST_DIR/$INTERP/InfoMgt/InfoGateway** directory.

#### Customizing Statements

To customize the sample configuration file statements, edit the file and modify the statements as required. Knowledge of the Tivoli Information Management for z/OS HLAPI and how it is used by remote applications is a prerequisite for editing and modifying the statements. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for a description of the HLAPI. References to other Tivoli Information Management for z/OS manuals are indicated when necessary.

The sample file has already been partially customized for you with suggested values that suit any application's system. You only need to customize a few statements in it that are related to your specific application's environment.

Customize the required values in the sample file that have been pre-filled with question mark (?) characters. After you are familiar with this file, you can customize any of the remaining statements.

#### General Syntax Rules

This is the general syntax when modifying the configuration file:

- Each statement in the configuration file must end with a semicolon. If the statement spans more than one line, only the last line in the statement should end with a semicolon.
- Comments can appear on any line in the file but must be preceded with double slash (//) characters.

## Process Steps

To configure Problem Service, perform the following tasks:

1. Customize the Problem Service sample configuration file:
   a. Customize the Tivoli Information Management for z/OS HLAPI information that Problem Service uses when establishing sessions with Tivoli Information Management for z/OS. See "Customizing the HLAPI Session Information" on page 271 for instructions.
   b. Customize the general Problem Service settings required for Problem Service operations. See "Customizing Problem Service General Settings" on page 275 for instructions.
   c. Customize the data mappings that Problem Service applies when transferring records back and forth between your applications and the Tivoli Information Management for z/OS database. See "Customizing Problem Service Data Mappings" on page 281 for instructions.
2. If you are using customized records on Tivoli Information Management for z/OS, prepare the HLAPI data views on MVS. See "Preparing the HLAPI Data Views on MVS" on page 293 for instructions.
3. Update the Services file. See "Updating the Services File" on page 295 for instructions.

You can customize the Problem Service configuration file at any time to modify existing settings. If your applications and Problem Service are running:
1. Stop your applications.
2. Stop Problem Service.
3. Modify the Problem Service configuration file.
4. Start Problem Service for the new configuration settings to take effect.
5. Start your applications.

# 21

# Customizing Your Problem Service Configuration File

This chapter explains how to plan for and configure the Problem Service option.

## Customizing the HLAPI Session Information

The configuration file contains a series of HLAPI-related statements that correspond to the HLAPI control PDB parameters. Problem Service uses these PDB parameters when establishing HLAPI sessions with Tivoli Information Management for z/OS and when performing transactions on records in the Tivoli Information Management for z/OS database. These statement values are used for all sessions started between Problem Service and Tivoli Information Management for z/OS.

The configuration file also contains HLAPI session statements that determine the number and the characteristics of the sessions that Problem Service will establish.

### HLAPI-Related Statements

This example shows the HLAPI-related statements in the sample configuration file.

```
// HLAPI-Related Statements

SessionMember="BLGSES??";
PrivilegeClass="?????";
TableCount=5;
APIMsgOption="B";
HLIMsgOption="B";
SpoolInterval=0;
TimeoutInterval=120;
DatabaseID="5";
DefaultOption="NONE";
DefaultDataStorageSize=1024;
SeparatorCharacter=",";
BypassPanel="NO";
ReplaceFreeformText=yes;
```

Customize the statements that appear in bold characters. These statements are required and are specific to your application's system. Ask your Tivoli Information Management for z/OS program administrator to provide you with these required values.

Table 38 on page 272 gives you a brief description of the HLAPI-related statements. As each of these statements corresponds to a specific parameter of the HLAPI control PDB, the correspondence with these parameters is also included in the table.

For more detailed descriptions of these parameters and the values they can assume, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

*Table 38. HLAPI Related-Statements for Sessions*

| Statement<br>PDB Parameter | Description |
|---|---|
| SessionMember<br><br>SESSION_MEMBER | This is a required statement. It specifies the name of the Tivoli Information Management for z/OS session-parameters member to be used for sessions with the HLAPI client.<br><br>The name can be up to 8 characters long, beginning with the character string **BLGSES**. It must match the name of the session member that the Tivoli Information Management for z/OS administrator defines in MVS. |
| PrivilegeClass<br><br>PRIVILEGE_CLASS | This is a required statement. It specifies the level of authority granted to users for performing certain operations on the Tivoli Information Management for z/OS database. Your Problem Service application is considered a Tivoli Information Management for z/OS user.<br><br>This value must match the name of the privilege class that the Tivoli Information Management for z/OS administrator assigns to your Problem Service application. The value can be 1 to 8 characters long. |
| TableCount<br><br>TABLE_COUNT | This is an optional statement. It specifies the number of Tivoli Information Management for z/OS tables that can be in storage during the session.<br><br>The value ranges from 0 to 256.<br><br>The default value is 0. The suggested value for Problem Service is 5. |
| APIMsgOption<br><br>APIMSG_OPTION | This is an optional statement. It specifies the destination of the low level application program interface (LLAPI) logging messages.<br><br>The valid values are P, C, or B. P indicates that LLAPI writes messages to the APIPRINT data set, and C indicates that the LLAPI messages will be put in message PDBs.<br><br>The default value is B, which indicates that LLAPI will perform both P and C.<br><br>This statement is used only if the SpoolInterval statement is specified and has a non-zero value. |
| HLIMsgOption<br><br>HLIMSG_OPTION | This is an optional statement. It specifies the destination of the HLAPI logging messages.<br><br>The valid values are P, C, or B. P indicates that HLAPI writes messages to the HLAPILOG data set, and C indicates that HLAPI puts the messages in the message PDBs.<br><br>The default value is B, which indicates that HLAPI will perform both P and C.<br><br>This statement is used only if the SpoolInterval statement is specified and has a non-zero value. |
| SpoolInterval<br><br>SPOOL_INTERVAL | This is an optional statement. It specifies the time interval, in minutes, for HLAPI and LLAPI logging in MVS.<br><br>The value ranges from 0 to 1440. The default value is 0, meaning that messages are not logged. In this case, the values in HLIMsgOption and APIMsgOption are ignored. |

*Table 38. HLAPI Related-Statements for Sessions (continued)*

| Statement / PDB Parameter | Description |
|---|---|
| TimeoutInterval<br><br>TIMEOUT_INTERVAL | This is an optional statement. It specifies the number of seconds that a database transaction can run before a timeout causes HLAPI to terminate the session.<br><br>The value ranges from 0 to 300, but if you specify a value between 0 and 45, the interval is set to 45 seconds.<br><br>The default value is 300 seconds. The suggested value is 120. |
| DatabaseID<br><br>DATABASE_ID | This is an optional statement. It specifies the name or ID number of the database that your application accesses during the session. For Tivoli Information Management for z/OS records that can be created or updated, the database ID is 5; do not change this value.<br><br>The default value is 5. |
| DefaultOption<br><br>DEFAULT_OPTION | This is an optional statement. It specifies which Tivoli Information Management for z/OS record fields are candidates for default data response processing by HLAPI during the session.<br><br>The valid values are:<br>■ REQUIRED<br>■ ALL<br>■ NONE<br><br>The default value is NONE. |
| DefaultDataStorageSize<br><br>DEFAULT_DATA_STORAGE_SIZE | This is an optional statement. It specifies the additional storage, in bytes, that HLAPI uses to hold the default response data.<br><br>The default value is 1024. |
| SeparatorCharacter<br><br>SEPARATOR_CHARACTER | This is an optional statement. It specifies the separator character that HLAPI uses to process response data. This character is used by HLAPI to separate the data items in a record field, if the field contains a list of data items.<br><br>The default value is the comma. |
| BypassPanel<br><br>BYPASS_PANEL_PROCESSING | This is an optional statement. It is used globally for all Tivoli Information Management for z/OS API sessions that this Problem Service starts. Only used on the API initialization transaction; applies for all of the transactions the gateway performs.<br><br>The default value is NO. |
| ReplaceFreeformText<br><br>REPLACE_TEXT_DATA | This is an optional statement. It is used globally for the Tivoli Information Management for z/OS API sessions that this Problem Service starts. It is used on an update transaction or a propagate transaction when propagating a record that has already been propagated (results in an update of the record). If it has a value of yes, the update uses the REPLACE_TEXT_DATA HLAPI control PDB with a value of YES. This indicates that new freeform text being supplied with the update replaces existing freeform text of the same type in the record being updated. Problem Service does not specify the REPLACE_FREEFORM_TEXT PDB if ReplaceFreeformText has a value of NO.<br><br>The default value is YES. |

## HLAPI Session Statements

The other HLAPI statements that you must customize in the configuration file are the HLAPI session statements. These statements determine:

- The number of sessions that Problem Service will establish with the Tivoli Information Management for z/OS HLAPI to perform Problem Service operations on the Tivoli Information Management for z/OS database.
- The characteristics of the physical connection to the MVS host where Tivoli Information Management for z/OS resides.
- The type of Problem Service operation for which Problem Service will use each session

The number of sessions is determined by the number of Session statements specified in the configuration file.

The following example shows the session statements as they appear in the Problem Service sample configuration file. Customize the keywords in bold characters.

```
// Session statements
Session Transaction=outbound
   DatabaseProfile="/usr/lpp/idbhlapi/examples/idbdb.pro"
   UserID="?????"   Password="?????";
Session Transaction=outbound
   DatabaseProfile="/usr/lpp/idbhlapi/examples/idbdb.pro"
   UserID="?????"   Password="?????";
Session Transaction=reverse
   DatabaseProfile="/usr/lpp/idbhlapi/examples/idbdb.pro"
   UserID="?????"   Password="?????";
Session Transaction=monitor
   DatabaseProfile="/usr/lpp/idbhlapi/examples/idbdb.pro"
   UserID="?????"   Password="?????";
```

Each session statement has four keywords: Transaction, DatabaseProfile, UserID, and Password.

**Transaction**

This keyword specifies the type of Problem Service operations for which the session will be used.

The values are:

**monitor**

Indicates monitor operations.

**outbound**

Indicates that the session will be used for all operations except monitor and reverse assignment.

**reverse**

Indicates reverse assignment operations.

You can specify only one session statement for each of the monitor and reverse assignment operations. You can specify up to 30 session statements for outbound operations. A separate daemon is started for each session.

You should specify at least one session for each type of transaction to obtain full Problem Service functionality. Otherwise, the Problem Service operation corresponding to an unspecified transaction will not be performed. For example, when you do not specify the reverse transaction in any of the session statements, Problem Service does not perform reverse assignment operations.

**DatabaseProfile**

The name of the database profile that you configured for the HLAPI client.

The database profile determines the connection characteristics for the session. If you are using the default database profile from the HLAPI client installation, use:

- For AIX:

```
/usr/lpp/idbhlapi/examples/idbdb.pro
```

- For Windows NT:

```
x:\infoapi\sample\database.pro
```

  where *x* is the drive on which the HLAPI client was installed.

If you are using a different database profile, you must specify the path and name of that file.

**Note:** Omit the path specification if the database profile resides in the current working directory for Problem Service.

Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for more information on the database profile usage and specification.

**UserID**

A user ID assigned for the MVS logon for the session.

This user ID is assigned to you by the MVS administrator and it must have the necessary authorizations to access the Tivoli Information Management for z/OS database. UserID corresponds to the SECURITY_ID parameter in the HLAPI control PDB used for establishing the session.

**Password**

The password assigned for the MVS logon for the session.

This parameter corresponds to the PASSWORD parameter in the HLAPI control PDB used for establishing the session.

By default, the HLAPI client requester will use a separate physical connection for each session you customize. If you want your sessions to share a single physical connection to the Tivoli Information Management for z/OS system, you must start the HLAPI client requester with a system profile. In the system profile, you must set parameter IDBSHARECMS equal to 1. Then your sessions will share a single physical connection if you specify the same values for UserID, Password, and DatabaseProfile in all the session statements.

Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for more information about the IDBSHARECMS parameter and the sharing of physical connections.

## Customizing Problem Service General Settings

The statements in the second part of the configuration file define general settings that control the operating characteristics of Problem Service. This example provides partially customized statements as they appear in the sample configuration file. You must customize the statements that appear in bold characters. If any of the remaining customized statements do not suit your application's system requirements, modify them.

```
// Problem Service General Settings
GatewayID="?????";
ReverseAssignInterval=60;
MonitorInterval=60;
InfoGatewayService="infogateway";
ForeignHost="?????";
RACallAPP="rassign";
MonCallApp="moninfo";
UnconditionalShutdownWait=10;
GatewayIDField="S1260";
GatewayIDPrefix="GWID/";
ConvertToUppercase=yes;
MaximumHits=1000;
MaxTextRetrieveLines=20;
IMPatternValidation=yes;
IMRNIDField="S0CCF";
IMCheckoutField="S14EF";
AssociatedDataField="S0E0F';
RetrieveBeforeUpdate=no;
SessionRetryInterval=10;
SessionRetryLimit=10;
InputJustChangedData=no;
PerformDataMapping=yes;
```

The following list gives a description of the statements in this part of the configuration file and explains how to customize them.

**GatewayID (Required)**

Defines the unique identifier for a Problem Service instance. It can be 1 to 8 characters long.

GatewayID corresponds to the APPLICATION_ID parameter in the HLAPI control PDBs. The ID you specify must be an eligible user ID defined in the same privilege class that you use in the **PrivilegeClass** statement in the configuration file.

Problem Service uses the GatewayID value for two main purposes:

- To identify itself as an eligible Tivoli Information Management for z/OS user, so that it can perform database record transactions during the HLAPI sessions with Tivoli Information Management for z/OS. The same GatewayID value is used for all the HLAPI sessions it establishes with Tivoli Information Management for z/OS.
- To identify all the Tivoli Information Management for z/OS records that it processes so that they can later be associated to this particular Problem Service application. The GatewayID value is stored in a field present in all Tivoli Information Management for z/OS records it processes. It also stores a prefix character, along with the GatewayID value, to indicate the operation performed on each of these records:

| Prefix | Definition |
|---|---|
| **P** | Propagated |
| **T** | Transferred |
| **R** | Reverse assigned |

When you customize the data mappings in the configuration file, specify the GatewayID value and prefix character in the propagate and transfer data mappings. For the records that your application reverse assigns, it must update the Tivoli Information Management for z/OS record's GatewayIDField with a reverse assignment gateway ID (R*GatewayID*). This keeps your application from being notified of the same Tivoli Information Management for z/OS records over and over again.

The Problem Service monitor operation uses this GatewayID value to identify all the records transferred and reverse assigned by this particular Problem Service.

**Note:** In a Tivoli Information Management for z/OS environment where more than one Problem Service or instance is present, you must make sure that the GatewayID is unique for each Problem Service. Otherwise, Problem Service operations will produce unpredictable results.

It is recommended that you use the GatewayID value as the checkout indicator value when customizing the data mappings for the propagate operation. This causes an automatic check-out of the records that are propagated by your application and enables you to update these records.

**Note:** After you are assigned a GatewayID value, you should not change it. However, if such a need arises, in addition to changing the value in the GatewayID statement, you must change all the occurrences of the GatewayID value in the rest of the Problem Service configuration file. Otherwise, unpredictable results will occur.

Furthermore, if you change the GatewayID value, all the records in Tivoli Information Management for z/OS that were previously processed by Problem Service will contain the old GatewayID value.

**ReverseAssignInterval (Optional)**
Specifies the time interval, in minutes, that Problem Service waits between each activation of the reverse assignment operation.

The value ranges from 0 to 999999.

The default value is 60 minutes.

**MonitorInterval (Optional)**
Specifies the time interval, in minutes, that Problem Service waits between each activation of the monitor operation.

The value ranges from 0 to 999999.

The default value is 60 minutes.

**InfoGatewayService (Required)**
Specifies the service name of the TCP/IP port used by Problem Service.

The sample configuration file provides the InfoGatewayService service name, which must be *infogateway*. Define this service name in the services file, and assign it an available TCP/IP port number.

**ForeignHost (Required)**
Specifies the name of the host that holds the executables specified by the RACallApp and MonCallApp statements in the configuration file. The specified host must be a managed node.

To determine the host name for AIX, issue the **hostname** AIX command from the AIX command line on the system that holds the executables.

To determine the hostname for a Windows NT system, on the Windows NT desktop select **My Computer** → **Control Panel**→ **Network**→ **Protocols** → **TCP/IP Protocol**→ **Properties** → **DNS**.

**RACallApp (Optional)**
Specifies the name of the executable to be invoked whenever the reverse assignment operation finds a search match in the Tivoli Information Management for z/OS database. The host name of the system where the executable resides is specified by the ForeignHost statement.

This executable should be written to accept three parameters: the Tivoli Information Management for z/OS record ID, its record type, and the reverse assignment gateway ID to be used to update the Tivoli Information Management for z/OS record when the application accepts the record.

**MonCallApp (Optional)**
Specifies the name of the executable to be invoked whenever the monitor operation finds a record in the Tivoli Information Management for z/OS database that belongs to this application's gateway ID and has been updated by Tivoli Information Management for z/OS or another application. The host name of the system where the executable resides is specified by the ForeignHost statement.

This executable should be written to accept two parameters: the Tivoli Information Management for z/OS record ID and its record type.

**UnconditionalShutdownWait (Optional)**
Specifies the number of seconds that Problem Service waits before sending unconditional shut down signals to terminate unclosed subprocesses, after having previously attempted to shut down those processes.

The value range is 5 to 300 seconds.

The default value is 10 seconds.

**GatewayIDField (Required)**
Specifies the index or alias name of a Tivoli Information Management for z/OS record field that will hold the GatewayID value. Problem Service needs such a field to store the GatewayID value and prefix character for all the records it propagates and transfers. For reverse assigns, you must store the GatewayID value in this field.

The sample configuration file sets GatewayIDField to s-word index S1260. The HLAPI PIDTs shipped with Tivoli Information Management for z/OS for problem and change records contain this index.

You can use this suggested Tivoli Information Management for z/OS field or use another field (either an existing one or another user-defined field) to hold the GatewayID value. If you choose to use another Tivoli Information Management for z/OS field, customize the create, update, and retrieve PIDTs and PIPTs or your data model records in the Tivoli Information Management for z/OS system on MVS to include this field. See "Preparing the HLAPI Data Views on MVS" on page 293 for information on customizing PIDTs and PIPTs.

**GatewayIDPrefix (Required)**
Specifies the p-word that is associated with the Tivoli Information Management for z/OS GatewayID field. It is used for searching the Tivoli Information Management for z/OS database for the records processed by Problem Service.

A p-word can contain up to 6 characters, where the last character must be a slash (/) or an underscore (_).

The suggested value provided in the sample configuration file is `GWID/`.

**ConvertToUppercase (Optional)**

Specifies whether or not to convert to uppercase characters all field data, except for freeform text data, that Problem Service stores in the Tivoli Information Management for z/OS database.

The possible values are:

| | |
|---|---|
| **yes** | Indicates that the data will be converted to uppercase characters. |
| **no** | Indicates that the data will not be converted. |

The default value is yes.

**MaximumHits (Optional)**

Corresponds to the NUMBER_OF_HITS parameter of the HLAPI control PDB. It is used for the HLAPI inquiry transaction on Tivoli Information Management for z/OS records. It specifies the maximum number of matches to be returned from a search. It applies to the Problem Service monitor, reverse assignment, and search operations.

The value ranges from 0 to 9999. The value 0 is treated as if the MaximumHits statement has not been specified.

The default value is 500. The suggested value for Problem Service is 1000.

**MaxTextRetrieveLines (Optional)**

Corresponds to the TEXT_UNITS parameter of the HLAPI control PDB. It specifies the maximum number of lines that can be retrieved for each freeform text field of a Tivoli Information Management for z/OS record.

The value ranges from 1 to 9999.

The default value is 1000. The suggested value for Problem Service is 20.

**IMPatternValidation (Optional)**

Specifies whether or not HLAPI should perform pattern validation on the input data fields for the records that Problem Service creates or updates in the Tivoli Information Management for z/OS database.

The possible values are:

| | |
|---|---|
| **yes** | Indicates that pattern validation will be performed. |
| **no** | Indicates that pattern validation will not be performed. |

The default value is yes.

**IMRNIDField (Required)**

Specifies the index or alias name of the Tivoli Information Management for z/OS record field that holds the record number identifier for the Tivoli Information Management for z/OS record.

For Tivoli Information Management for z/OS records, the s-word index for the default RNID field is S0CCF.

**IMCheckoutField (Required)**

Specifies the index or alias name of the Tivoli Information Management for z/OS record field that is used as the indicator that the record is locked or checked out in Tivoli Information Management for z/OS.

The Tivoli Information Management for z/OS records already contain a field that Tivoli Information Management for z/OS uses to indicate that records are checked out. The s-word index of this field is S14EF, which you must specify for this statement.

The HLAPI PIDTs shipped with Tivoli Information Management for z/OS for problem and change records contain this index. If you have customized PIDTs or data model records, you must add this index to those records so that Problem Service can use this field for propagated records. See “Preparing the HLAPI Data Views on MVS” on page 293 for information on customizing PIDTs.

**Note:** Specify this field in the propagate data mappings, in the third part of the Problem Service configuration file, so that Problem Service automatically checks out propagated records.

**AssociatedDataField (Required)**

Specifies the index or alias of the Tivoli Information Management for z/OS field that is to be returned for matches on the search operation.

**RetrieveBeforeUpdate (Optional)**

Specifies whether Problem Service needs to perform special processing before updating a record in the Tivoli Information Management for z/OS database. This statement is applicable when you are using record fields that contain a Tivoli Information Management for z/OS List Processor list of data items. It also affects how Problem Service handles the deletion of null fields when updating the Tivoli Information Management for z/OS record.

The possible values are:

**yes** Indicates that Problem Service will first retrieve the record fields from the Tivoli Information Management for z/OS database and then update them.

**Note:** The retrieve is not performed if data mapping is disabled.

**no** Indicates that Problem Service will update the record fields directly, without retrieving them.

The default value is yes. The suggested value for Problem Service is no.

If the record is retrieved before the update, Problem Service can verify the fields that are already null and does not need to perform extra delete processing to delete their contents. If the record is not retrieved, Problem Service attempts to delete the field contents even if the fields are already empty. This has an impact on performance if the record contains a large number of empty fields.

When updating a Tivoli Information Management for z/OS List Processor list with a shortened list, you can specify **yes** to ensure the Tivoli Information Management for z/OS field is updated correctly. If you specify **no**, you must explicitly delete unwanted list entries by using the HLAPI separator character.

**SessionRetryInterval (Required)**
Specifies the time interval, in minutes, that Problem Service will delay between attempts to start a session that has stopped for some reason other than being shut down.

The maximum value for the interval is 30 minutes.

**SessionRetryLimit (Required)**
Specifies the number of times that Problem Service will attempt to restart a stopped session.

The maximum value for the number of retries is 10. A value of 0 directs Problem Service not to attempt session restarts.

**Note:** It is recommended that you initially set this value to zero (0). After Problem Service is installed and configured, a higher value can be set.

**InputJustChangedData (Optional)**
Specifies whether the caller will provide all data for a record or just the changed (delta) data. This is important when performing a Tivoli Information Management for z/OS update (updating or propagating) to an existing record.

**yes** Problem Service attempts to delete from the Tivoli Information Management for z/OS record null fields passed by the caller. Special list processor field processing is not performed. The caller must ensure that existing entries in a list to be updated are deleted if necessary. This is important when passing a parameter list that has fewer entries than the current list it is updating.

**no** Problem Service attempts to delete Tivoli Information Management for z/OS fields that are defined in your data mappings, but were not passed by your application or hardcoded in the data mappings.

The default value is no.

**PerformDataMapping (Optional)**
The possible values are:

**yes** The data mapping function of the gateway is used.

**no** Data mapping is not performed; all data is in Tivoli Information Management for z/OS format. If updating a Tivoli Information Management for z/OS record, optional record retrieval and null field deletion is not performed, unless you are only passing changed (delta) data.

The default value is yes.

## Customizing Problem Service Data Mappings

The main purpose of the third part of the configuration file is to define the data mappings that will be applied during Problem Service operations and Tivoli Information Management for z/OS transactions that share data between Tivoli Information Management for z/OS and your application's records. If you are an end user of Problem Service, refer to your application's documentation for help in mapping records to Tivoli Information Management for z/OS.

This part of the configuration file also contains statements that define the data required by the Tivoli Information Management for z/OS HLAPI transactions as well as statements that enable you to customize the search criteria for the reverse assignment operation. You must complete the customization of this part of the configuration file before you can use Problem Service.

The data mappings in the sample configuration file have been customized to best fit a sample application's requirements and to find the best match between the Tivoli Information Management for z/OS database fields and the application's record fields. The mappings are based on uncustomized Tivoli Information Management for z/OS records. If they have been customized, the data mappings need to be adjusted accordingly. You can disable Problem Service data mapping by specifying:

```
PerformDataMapping=no
```

## Mapping Your Application and Tivoli Information Management for z/OS Records

Fields within mapped records that are contained in the Tivoli Information Management for z/OS record and your application's record will be mapped. Remember that the mappings in the sample configuration file are based on uncustomized Tivoli Information Management for z/OS records. You can configure the mapping to:

- Alter a mapped-to Tivoli Information Management for z/OS record field.
- Add a new mapped-to Tivoli Information Management for z/OS record field.
- Remove a mapped-to Tivoli Information Management for z/OS record field.

Tivoli Information Management for z/OS list processor and multiple response fields are supported. List processor fields contain from 1 to 19274 entries. An example is a name field that allows a first and last name separated by a blank. Most Tivoli Information Management for z/OS fields are single response fields.

HLAPI requires that lists and multiple responses for a field be entered as separate strings with each response separated by a separator character (default is a comma). The data for these fields must be passed to Problem Service containing these separator characters or must be manipulated by data mappings to include them. For example, the data can be separated by blanks when passed to Problem Service and the mappings can convert these blanks into the separator character.

The default mappings supplied in the sample configuration file do not contain list processor fields. The only Tivoli Information Management for z/OS fields that are multiple response fields are those customized by the user.

See "Supported Data Conversions" on page 307 for information on converting data and for a list of the data conversions supplied with Problem Service.

### Setting Up the Data Mapping Rules

Fields can be mapped between your application's records and Tivoli Information Management for z/OS records. Using mapping statements in the configuration file, data is mapped to produce a collection of data that has Tivoli Information Management for z/OS data keys and Tivoli Information Management for z/OS format.

For transactions, such as propagate and transfer, that put a record into the Tivoli Information Management for z/OS database, a mapping statement with a hardcoded gateway ID (with prefix) should be included. The prefix indicates whether the Tivoli Information Management for z/OS record has been propagated or transferred.

**Prefix Definition**

| | |
|---|---|
| **P** | Record has been propagated. |
| **T** | Record has been transferred. |

In the default mappings, the gateway ID is identified by the Tivoli Information Management for z/OS s-word index S1260.

Table 39 shows the uncustomized Tivoli Information Management for z/OS record fields to which the sample application record fields are mapped by the Problem Service samples.

*Table 39. Uncustomized Tivoli Information Management for z/OS and Sample Application Record Fields and Their Attributes*

| **Uncustomized Tivoli Information Management for z/OS Field** | | | **Sample Application Field** | |
|---|---|---|---|---|
| **Name** | **Index** | **Attributes** | **Name** | **Attributes** |
| Assignee department | S0B9C | 11 alphanumeric characters including #, @, $, &, or / | Organization | 30 characters, varchar |
| Assignee name | S0B5A | 1–15 alphanumeric characters including #, @, $, &, or / | Assignee | 90 characters, varchar |
| Current priority | S0BE7 | 1–2 numeric | Priority | 1 digit (1, 2, 3, 4, 5) or None (required) |
| Date/Time opened | S0C3E or S0C74 | Date: external date format; Time: external time format | StartDate | MM/DD/YY(YY) hh:mm:ss (a│p) |
| Description abstract | S0E0F | 1–45 freeform (string) | (not mapped) | |
| Description text | S0E01 | Freeform text | Description | 240 characters, freeform text |
| Gateway ID | S1260 | 1–8 alphanumeric characters including #, @, $, &, or / | (not mapped) | |
| Problem status | S0BEE | INITIAL OPEN CLOSED | Status | Approved Closed Open Working Pending Rejected Complete |
| Problem type | S0C09 | 1–8 alphameric characters including #, @, $, &, or / | TroubleCode (hierarchy classifying the problem) | Length is 40, data can include '.' |
| Record ID | S0CCF | 1–8 alphanumeric characters including #, @, $, &, or / | rnid | 30 characters, varchar |
| Reported by | S0B59 | 1–15 alphanumeric characters including #, @, $, &, or / | Originator | 90 characters, varchar |
| Status text | S0E02 | Freeform text | Detail | 1000 characters, freeform text |
| Tracked by | S0B5C | 1–15 alphanumeric characters including #, @, $, &, or / | Modifier | 90 characters (first, middle, last) |
| Vendor PMR number | S0F52 | 8 numeric | TicketNum | 8 numeric |

## Mapping Fields to Problem Service Operations

Not all fields are mapped for each operation being performed. In the configuration file, specify the fields that are to be mapped for each operation by using transactions statements. They enable fields to be mapped to one or more Problem Service operations. Different fields can be mapped to different operations.

Data mapping statements specified before the first occurrence of a transactions statement apply to all operations. Otherwise, the data mapping statements apply only to the operations specified by the most recent transactions statement. For example:

```
Transactions=propagate,transfer,update;
 S0B5A(15)<<translate(Assignee,", ","//");
```

In this example, the data mapping statement following the transactions statement applies to the Problem Service operations: propagate, transfer, and update.

Multiple transactions statements can be specified. When multiple transactions statements are specified with the same keyword, the mapping definitions for each occurrence are grouped together for that operation; the last occurrence does not override previous occurrences.

## Understanding the Syntax of Data Mapping Statements

Data mapping statements identify the data to be mapped, how to transform the data, and where to put the data. The direction of the mapping is indicated by double less than symbols (<<) or double greater than symbols (>>):

**<<** Specifies that the mapping applies to data flowing from your application's database to the Tivoli Information Management for z/OS database. The target of the mapping is the operand to the left of the <<.

**>>** Specifies that the mapping applies to data flowing from the Tivoli Information Management for z/OS database to your application's database. The target of the mapping is the operand to the right of the >>.

The mapping target operand is a field name, suffixed by a length enclosed in parentheses. Field names appearing to the left of the << or >> are assumed to be Tivoli Information Management for z/OS field names, and field names appearing to the right of the << or >> are assumed to be your application's field names. A Tivoli Information Management for z/OS field name can be one of the following:

- An alias name as defined in a program alias table (PALT).
- An S followed by an s-word index as defined in the PIDT field PIDTSYMB.
- A P followed by a prefix index as defined in the PIDT field PIDTSYMB.

The source for the mapping is a combination of literal strings, field names, and user exit specifications. Data associated with source field names, and data returned from user exit calls, is substituted by the mapping facility. User exits are specified as user exit name, followed by a comma delimited argument list enclosed in parentheses. Each of these arguments can be a combination of integers, literal strings, field names, and user exit specifications. See “Supported Data Conversions” on page 307 for information on data conversions and user exits.

The syntax for a mapping into a Tivoli Information Management for z/OS field is as follows:

```
IMFieldName(length) << literal_string                 ;
                       subroutineName(parm1, parm2, ...)
                       your_application's_FieldName
                       [combination of the above]
```

Where `parm`*n* can be:

```
  integer
  literal_string
  your_application's_FieldName
  subroutineName(parm1, parm2, ...)
```

The following is an example of a mapping into your application's field:

```
literal_string      >> your_application's_FieldName(length);
subroutineName(parm1, parm2, ...)
IMFieldName
[combination of the above]
```

Where `parm`*n* can be:

```
  integer
  literal_string
  IMFieldName
  subroutineName(parm1, parm2, ...)
```

### Syntax Examples

The following are some examples of data mappings in both directions:

- `S0E0F<<"This is a description abstract.";`

  For appropriate transaction types (for example, transfer), the Tivoli Information Management for z/OS field defined in the PIDT as S0E0F is assigned the value `This is a description abstract.`

- `S0E02>>detail(1000);`

  For appropriate transaction types (for example, retrieve), the data in the Tivoli Information Management for z/OS field defined in the PIDT as S0E02 is assigned to your application's field named detail. The target field is truncated or padded with blanks to 1000 bytes.

- `"@"S0B59>>Originator;`

  For appropriate transaction types (for example, retrieve), an @ is added as a prefix to the data in the Tivoli Information Management for z/OS field defined in the PIDT as S0B59. The result of this operation is then assigned to your application's field named Originator.

## Changing the Data Mapping Rules

You can change data mappings rules by adding, modifying, or deleting mapping statements in the configuration file.

### Adding a Field

When a new field is added, a new mapping statement must be added to the configuration file if the field is to be shared with Tivoli Information Management for z/OS. All necessary conversions must be specified here. The Tivoli Information Management for z/OS PIDTs and PIPTs must be rebuilt or data model records modified to allow Problem Service to process the new field in Tivoli Information Management for z/OS.

### Changing a Field

If a field in your application's record is changed, it might be necessary to change the mapping statements by:

- Converting it in a different way
- Mapping it into a different Tivoli Information Management for z/OS field
- Changing the operating characteristics of Problem Service

Tivoli Information Management for z/OS PIDTs and PIPTs might have to be rebuilt to allow the Tivoli Information Management for z/OS HLAPI to correctly validate data sent by Problem Service. Keep the Problem Service and Tivoli Information Management for z/OS data models as similar as possible to minimize the data conversions needed to share data between the two databases.

### Removing a Field

When a field in your application's record is removed, check the configuration file to ensure that there are no mapping statements mapping Tivoli Information Management for z/OS fields into the removed field. If there are, these mapping statements need to be removed from the configuration file.

## Sample Configuration File Descriptions

In the sample configuration file, values have been included that you need not change for a basic configuration. It assumes that Tivoli Information Management for z/OS records have not been customized. Values that you **must** provide are clearly indicated by bold characters or question mark (?) characters in the following descriptions.

Data mappings should be defined for all fields that are to be shared between your application and Tivoli Information Management for z/OS records.

### Defining Specific Record Types

The part of the configuration file shown in this example contains statements needed by the Tivoli Information Management for z/OS HLAPI. The statement descriptions are:

```
//******************************************************************************
//
// Define information to use for specific record types including data
// required by the Information Management for z/OS HLAPI and data mappings for
// data sharing between the local application and Information Management for z/OS.
//
//******************************************************************************

//******************************************************************************
//
// Record Type:  Helpdeskapp
//
// Define information for a Helpdeskapp record.  Helpdeskapp
// records map to Information Management for z/OS records.
//
//******************************************************************************
RecordType

    // ---------------------------------------------------------------------
    // String that identifies the record type.
    // ---------------------------------------------------------------------
    RecordTypeValue="Helpdeskapp"
    // ---------------------------------------------------------------------
    // Key of field in helpdesk record that contains the
    // Information Management for z/OS RNID.
    // ---------------------------------------------------------------------
    ForeignIMRNIDField="rnid"
```

```
// ----------------------------------------------------------------------
// Key of field (PIDTSYMB value or alias name) that contains the identifier
// of the helpdesk record - only used by Reverse Assignment and Monitor.
// ----------------------------------------------------------------------
IMForeignRNIDField="S0F52"
```

**RecordTypeValue="Helpdeskapp"**
The string that identifies the record type. Identify the record type as the appropriate record type for your application.

**ForeignIMRNIDField="rnid"**
The name of the field in your application record that contains the Tivoli Information Management for z/OS identifier.

**IMForeignRNIDField="S0F52"**
The Tivoli Information Management for z/OS field (PIDTSYMB value or alias name) that contains the identifier of your application record. This is only used by the reverse assignment and monitor operations.

Another value that can be defined, but which has not been included in the sample configuration file, is:

**AliasTable**
This defines the value for the HLAPI control PDB ALIAS_TABLE. When this statement is specified, the ALIAS_TABLE control PDB is specified for all Tivoli Information Management for z/OS HLAPI create, update, retrieve, and inquiry transactions performed for this record type.

## Defining API PIDT Names

The required Tivoli Information Management for z/OS API PIDT names are specified by the following configuration file statements:

```
CreateDataView="P:BLGYPRC"
UpdateDataView="P:BLGYPRU"
DisplayDataView="P:BLGYPRR"
SearchDataView="P:BLGYPRI";
```

They specify the set of Tivoli Information Management for z/OS PIDTs or data view record names to be used with the record type field value (see the RecordTypeValue keyword). These statements can be specified more than one time in the configuration file, but are locally mutually inclusive (all must appear wherever one appears). These statements must immediately follow a RecordTypeValue statement.

The data value for these PIDT statements must begin with P: or D:. P: indicates that the rest of the value is a PIDT name (for example, P:BLGYPRC). D: indicates that the rest of the value is a data view record ID.

**CreateDataView="P:BLGYPRC"**
This is the name of the PIDT that creates Tivoli Information Management for z/OS problem records.

**UpdateDataView="P:BLGYPRU"**
This is the name of the PIDT that updates Tivoli Information Management for z/OS problem records.

**DisplayDataView="P:BLGYPRR"**
This is the name of the PIDT that retrieves Tivoli Information Management for z/OS problem records.

**SearchDataView="P:BLGYPRI"**
This is the name of the PIDT table used to search for Tivoli Information Management for z/OS problem records.

## Defining Freeform Text Fields

The following sample configuration file statement defines the Tivoli Information Management for z/OS fields that are freeform text together with their associated line widths:

```
 IMText Width=60 Fields="S0E01", "S0E02";
```

In the example, the Tivoli Information Management for z/OS fields S0E01 and S0E02 are freeform text fields with a line width of 60 characters.

The application's record text is broken up into segments using the width value with each segment becoming a line of text in the Tivoli Information Management for z/OS record. The Problem Service transfer, propagate, and update operations use the value specified for width to break up the application's input data into Tivoli Information Management for z/OS freeform text lines of this width. The retrieve operation uses the maximum width for the record type as the width to retrieve, and this value then becomes the value of the HLAPI control PDB TEXT_UNITS.

Tivoli Information Management for z/OS freeform text lines are truncated or padded with blanks to the width specified.

## Defining the ReverseArguments Statement

One or more structured or freeform search arguments can be specified for the reverse assignment operation. If no arguments are specified, reverse assignment will not be performed. The following is a sample configuration file ReverseArguments statement:

```
ReverseArguments S0B5A="??????";
```

For example, when **SMITH** replaces **??????** in the sample, it becomes:

```
ReverseArguments S0B5A="SMITH";
```

This means that all records with the field S0B5A containing the value SMITH will be reverse assigned. This is an example of a structured search argument. If a search argument is specified by itself, without an associated field name, it is a freeform search.

If more than one search argument is specified, all conditions must be satisfied for the record to be reverse assigned. Arguments can include a Boolean operator as the first character (valid operators are those allowed by the Tivoli Information Management for z/OS HLAPI for freeform search arguments). In the following example, all records for which the S0B5A field contains SMITH and not reported by JONES will be reverse assigned.

```
ReverseArguments S0B5A="SMITH" "¬PERS/JONES";
```

Reverse assignment can act against records transferred by other gateways, but not against records transferred by this gateway. When a transferred record is reverse assigned, the originating Problem Service stops monitoring it for changes. Reverse assignment does not act against records reverse assigned by another gateway. It checks the value in the GatewayID field. It also ignores records that have been checked out.

ReverseArguments constructs the total search argument using the following information:

- ReverseArguments structured arguments

- Predefined arguments (for example, not transferred by this gateway)
- ReverseArguments freeform arguments

## Mapping Records from Your Application to Tivoli Information Management for z/OS

The mappings as shown in this example are defined for the Problem Service propagate, transfer, and update operations as specified in the transactions statement. Your application's data is mapped to Tivoli Information Management for z/OS data through the use of user exits. For information on these user exits, refer to "Specifying User Exits for Conversions" on page 309.

```
Transactions=propagate,transfer,update;
 S0B5A(15)<<translate(Assignee,", ","//");
 S0B9C(11)<<translate(Organization,", ","//");
 S0B5C(15)<<translate(Modifier,", ","//");
 S0BE7<<change(Priority,"None","");
 S0C3E<<(words(StartDate,1,1));
 S0C74<<toIMTime(words(StartDate,2));
 S0BEE<<translateWord(Status,"Approved","OPEN",
                             "Pending","OPEN",
                             "Working","OPEN",
                             "Complete","CLOSED",
                             "Rejected","CLOSED");
 S0C09(8)<<translate(TroubleCode,".","/");
 S0B59(15)<<nullDefault(translate(Originator,", ","//"),
                             "Helpdeskapp") ;
```

**Where**:

**S0B5A(15)<<translate(Assignee,", ","//");**
All commas and spaces found within the value in the application's **Assignee** field are converted to slashes and the value is truncated to 15 characters.

**S0B9C(11)<<translate(Organization,", ","//");**
All commas and spaces found within the value in the application's **Organization** field are converted to slashes and the value is truncated to 11 characters.

**S0B5C(15)<<translate(Modifier,", ","//");**
All commas and spaces found within the value in the application's **Modifier** field are converted to slashes and the value is truncated to 15 characters.

**S0BE7<<change(Priority,"None","");**
When the application's **Priority** field contains the value None, this is converted to a null string.

**S0C3E<<toIMDate(StartDate,1,1);**
The first value in the application's **StartDate** field is a date. In this example, the format is mm/dd/yyyy.

**S0C74<<toIMTime(words(StartDate,2));**
This converts the second value (time) in the application's **StartDate** field from the format hh:mm:ss am or hh:mm:ss pm to the format hh:mm, by dropping the seconds.

**S0BEE<<translateWord(Status,"Approved","OPEN", "Pending","OPEN", "Working","OPEN", "Complete","CLOSED", "Rejected","CLOSED");**

- If the value in the application's **Status** field is **Approved**, **Pending**, or **Working**, convert it to **OPEN**.

- If the value in the application’s **Status** field is **Complete** or **Rejected**, convert it to **CLOSED**.

**S0C09(8)<<translate(TroubleCode,".","/");**
All periods found within the value in the application’s **TroubleCode** field are converted to slashes and the value is truncated to 8 characters.

**S0B59(15)<<nullDefault(translate(Originator,", ","//"), "Helpdeskapp");**
All commas and spaces found within the value in your application’s **Originator** field are converted to slashes and the value is truncated to 15 characters. If the Originator field is empty, a value of Helpdeskapp is mapped.

These mappings are defined for propagate and transfer as specified in the transactions statement. No conversions are performed on the application’s fields before being mapped.

```
Transactions=propagate,transfer;
 S0E01<<Description;
 00E02<<Detail;
 S0F52<<TicketNum;
```

The following mappings are defined for propagate as specified in the transactions statement.

```
Transactions=propagate;
 S0E0F<<"PROPAGATED RECORD FROM HELPDESKAPP";
 S1260<<"P?????";
 // ????? = GatewayID. Indicates this record was propagated
 // by this gateway.
 S14EF<<"?????";
 // ????? = GatewayID. Check-out indicator
```

**Where**:

**S0E0F<<"PROPAGATED RECORD FROM HELPDESKAPP";**
"PROPAGATED RECORD FROM HELPDESKAPP" is hardcoded into the Tivoli Information Management for z/OS field.

**S1260<<"P?????";**
Indicates this record was propagated by the gateway identified by the GatewayID value specified in substitution for P?????.

**S14EF<<"?????";**
Is the checkout indicator; the record has been checked out by the gateway indicated in substitution for ?????. By being checked out, propagated records can only be updated by this gateway.

The following mappings are defined for the transfer operation as specified in the transactions statement. In this case no conversions are made before the mappings.

```
Transactions=transfer;
 S0E0F<<"TRANSFERRED RECORD FROM HELPDESKAPP";
 S1260<<"T?????";
```

**Where**:

**S0E0F<<"TRANSFERRED RECORD FROM HELPDESKAPP";**
"TRANSFERRED RECORD FROM HELPDESKAPP" is hardcoded into the Tivoli Information Management for z/OS field.

**S1260<<"T?????";**
Indicates that the record was transferred by the gateway identified by the GatewayID value specified in substitution for T?????.

## Mapping Records from Tivoli Information Management for z/OS to Your Application

The following mappings are defined for the Retrieve transaction as specified in the transactions statement. Tivoli Information Management for z/OS data is mapped to your application's data.

```
Transactions=retrieve;
translateWord(change(nullDefault(S0B5A,"None"),"//",", ","/",","),
                 "NONE","None")>>Assignee;
S0E01>>Description(240);
translateWord(translate(nullDefault(S0B9C,"None"),"/"," "),
                 "ORGANIZATIO", "Organization",
                 "NONE","None")>>Organization;
nullDefault(stripLeading(fromIMPriority(S0BE7),"0"),"None")>>Priority;
translateWord(change(S0B59,"//",", ","/",", "),
                 "NONE","None")>>Originator;
translateWord(S0BEE,"INITIAL","Open",
                 "OPEN","Open", "CLOSED","Closed")>>Status;
translateWord(change(nullDefault(S0B5C,"None"),"//",", ","/",", "),
                 "NONE","None")>>Modifier;
translateWord(nullDefault(S0C09,"Unknown"),
                 "APPLICAT","Applications", "HARDWARE","Hardware",
                 "NETWORKS","Networks", "SOFTWARE","Software",
                 "UNKNOWN","Unknown") >> TroubleCode;
S0E02>>Detail(1000);
S0CA9>>Resource;
S0C3E" "translateWord(S0C74,"","","*",S0C74":00")>>StartDate;
S0CCF>>rnid;
```

**Where**:

**translateWord(change(nullDefault(S0B5A,"None"), "//",",","/",","), "NONE","None")>>Assignee;**
When the **S0B5A** field is empty, a value of "None" is mapped. If its value is "NONE", it is converted to "None". All slashes are converted to commas before being mapped into the application's **Assignee** field.

**S0E01>>Description(240);**
The first 240 characters of the Tivoli Information Management for z/OS **S0E01** field are mapped into the application's **Description** field.

**translateWord(translate(nullDefault(S0B9C,"None"),"/"," "), "ORGANIZATIO","Organization","NONE","None") >>Organization;**
When the **S0B9C** field is empty, a value of "None" is mapped. "ORGANIZATIO" is converted to "Organization" and "NONE" is converted to "None". All slashes are converted to spaces before being mapped into the application's **Organization** field.

**nullDefault(stripLeading(fromIMPriority(S0BE7),"0"),"None") >>Priority;**
When the value in the **S0BE7** field is between 0 and 5, it remains unaltered. Otherwise a value of "5" is mapped. Leading zeros are removed and if the field is empty, the value "None" is mapped into the application's **Priority** field.

**translateWord(change(S0B59,"//",", ","/",", "), "NONE","None")>>Originator;**
If the value of the **S0B59** field is ″NONE″, it is converted to ″None″. All slashes are converted to commas before being mapped into the application's **Originator** field.

**translateWord(S0BEE,"INITIAL","Open", "OPEN","Open", "CLOSED","Closed")>>Status;**
The value ″INITIAL″ in the Tivoli Information Management for z/OS **S0BEE** field

is changed to ″Open″, the value ″OPEN″ to ″Open″, and the value ″CLOSED″ to ″Closed″ before being mapped into the application's **Status** field.

**translateWord(change(nullDefault(S0B5C,"None"), "//",",","/", ","), "NONE","None") >>Modifier;**
When the **S0B5C** field is empty, a value of ″None″ is mapped. If its value is ″NONE″, it is converted to ″None″. All slashes are converted to commas before being mapped into the application's **Modifier** field.

**translateWord(nullDefault(S0C09,"Unknown"), ″APPLICAT″,″Applications″, ″HARDWARE″,″Hardware″, ″NETWORKS″,″Networks″, ″SOFTWARE″,″Software″, ″UNKNOWN″,″Unknown″) >>TroubleCode;**
When there is a null value in the Tivoli Information Management for z/OS **S0C09** field, it is replaced with the hardcoded value of ″Unknown″. All occurrences of ″APPLICAT″ in the field are replaced by ″Applications″, ″HARDWARE″ is replaced with ″Hardware, ″NETWORKS″ with ″Networks″, ″SOFTWARE″ with ″Software″, and ″UNKNOWN″ with ″Unknown″.

**S0E02>>Detail(1000);**
The first 1000 characters of the Tivoli Information Management for z/OS **S0E02** field are mapped into the application's **Detail** field

**S0CA9>>Resource;**
The value in the **S0CA9** field is mapped directly into the application's **Resource** field without being converted.

**S0C3E" "translateWord(S0C74,"", "","*",S0C74":00")>>StartDate;**
When the Tivoli Information Management for z/OS time field (**S0C74**) is empty, it remains empty. Otherwise, seconds are added (:00) to the time already specified in hours and minutes (hh:mm). The Tivoli Information Management for z/OS date field (**S0C3E**) and the time are concatenated before being mapped into the application's **StartDate** field. In this example, the date in the S0C3E field is in mm/dd/yyyy format.

**S0CCF>>rnid;**
The value in the **S0CCF** field is mapped directly into the application's **rnid** field without being converted.

# 22 Completing Problem Service Configuration

This chapter explains how to complete the configuration of Problem Service.

## Preparing the HLAPI Data Views on MVS

To define all the Tivoli Information Management for z/OS record fields that Problem Service needs to access when performing the Problem Service operations, you need HLAPI PIDTs and PIPTs or Tivoli Information Management for z/OS data model records.

The HLAPI PIDTs that you need for the database records are create, update, and retrieve. Problem Service requires two fields to be defined in the HLAPI PIDTs or the data model's data view records:

- The create, update, and retrieve PIDTs or data view records must contain the Tivoli Information Management for z/OS field that you specify in the GatewayIDField statement in the Problem Service configuration file. This field must be present in all Tivoli Information Management for z/OS records processed by Problem Service.
- The create PIDT or data view record must also contain the Tivoli Information Management for z/OS field specified in the IMCheckoutField statement in the Problem Service configuration. This is necessary for the propagate operation to always check out the propagated records.

How to proceed depends on whether the Tivoli Information Management for z/OS records you are using with Problem Service are:

- Uncustomized (see "Using PIDTs and PIPTs with Uncustomized Records" for information on the HLAPI tables shipped with Tivoli Information Management for z/OS.)
- Customized (see "Preparing PIDTs and PIPTs for Customized Records" on page 294 for information to assist you in preparing the HLAPI tables.)

To prepare the HLAPI tables on MVS you need the assistance of the Tivoli Information Management for z/OS administrator.

### Using PIDTs and PIPTs with Uncustomized Records

The HLAPI PIDTs and PIPTs shipped with Tivoli Information Management for z/OS for problem and change records have been built to reflect the Tivoli Information Management for z/OS field specifications used in the sample Problem Service configuration file. These tables contain the fields of uncustomized Tivoli Information Management for z/OS records as well as the two Tivoli Information Management for z/OS fields (GatewayIDField and IMCheckoutField) required by Problem Service.

The PIDTs and PIPTs shipped with Tivoli Information Management for z/OS are listed in Table 40.

*Table 40. Tivoli Information Management for z/OS PIDTs and PIPTs for Uncustomized Records*

| Filename | Description |
|---|---|
| BLGYPRC | Problem Record Create PIDT |
| BLGYPRCP | Problem Record Create PIPT |
| BLGYPRI | Problem Record Inquiry PIDT |
| BLGYPRIP | Problem Record Inquiry PIPT |
| BLGYPRR | Problem Record Retrieve PIDT |
| BLGYPRRP | Problem Record Retrieve PIPT |
| BLGYPRU | Problem Record Update PIDT |
| BLGYPRUP | Problem Record Update PIPT |
| BLGYCHC | Change Record Create PIDT |
| BLGYCHCP | Change Record Create PIPT |
| BLGYCHI | Change Record Inquiry PIDT |
| BLGYCHIP | Change Record Inquiry PIPT |
| BLGYCHR | Change Record Retrieve PIDT |
| BLGYCHRP | Change Record Retrieve PIPT |
| BLGYCHU | Change Record Update PIDT |
| BLGYCHUP | Change Record Update PIPT |

For more information on how Tivoli Information Management for z/OS uses PIDTs and PIPTs, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

## Preparing PIDTs and PIPTs for Customized Records

Use the MVS system where Tivoli Information Management for z/OS is running to create new PIDTs and PIPTs. Customizing these tables requires experience with the Tivoli Information Management for z/OS product and knowledge of how records are customized in Tivoli Information Management for z/OS. If you do not have experience with this product, you need the direct assistance of the Tivoli Information Management for z/OS administrator to customize these tables on MVS. The Tivoli Information Management for z/OS administrator will also have the necessary information about the characteristics of the fields of the customized records in Tivoli Information Management for z/OS.

Tivoli Information Management for z/OS provides you with an MVS utility, called **BLGUT8**, that you use to build each customized PIDT and corresponding PIPT in Tivoli Information Management for z/OS. The **BLGUT8** utility requires the following input:

- Control statements that specify the record fields for the PIDT.
- Tivoli Information Management for z/OS assisted entry panels (in offloaded format using utility **BLGUT6F**) for the fields specified in the PIDT.
- Tivoli Information Management for z/OS dictionary entries defining the characteristics of the fields specified in the PIDT.

**Note:** For more information on assisted entry panels and the Tivoli Information Management for z/OS dictionary, refer to the *Tivoli Information Management for z/OS Panel Modification Facility Guide*

The **BLGUT8** utility generates PIDTs and PIPTs that HLAPI uses with your application. For detailed descriptions on customizing HLAPI tables, using data model records, and using the **BLGUT8** utility, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

Shown below are the input statements that must be added to the BLGUT8 job stream that builds your customized HLAPI tables. These statements use the dictionary entries and assisted entry panels shipped with Tivoli Information Management for z/OS. If you use a different s-word index for the Gateway ID field, then you must update these statements to match your environment.

```
FIELD PANEL(BLM6GWID) INDEX(S1260);      /* GATEWAY ID FIELD */
FIELD PANEL(BLG6APPL) INDEX(S14EF);      /* CHECKOUT FIELD   */
```

## Updating the Services File

After you complete the customization of the Problem Service configuration file, update the services file on your workstation to reflect the settings you specified in the configuration file.

### AIX Workstation /etc/services File

In the **/etc/services** file, define the entry for the service name you specified in the InfoGatewayService statement and associate an available TCP/IP port number for this service name.

For example, if you customized the statement with the following service name:

```
InfoGatewayService="infogateway";
```

The entry in the **/etc/services** file would be as follows:

```
infogateway 1453/tcp
```

Select a TCP/IP port number (for example, *1453*) for the service name that is unique and available on your system.

### Windows NT Workstation Services File

In your IP services file, define the entry for the service name you specified in the InfoGatewayService statement and associate an available TCP/IP port number for this service name.

For example, if you customized the statement with the following service name:

```
InfoGatewayService="infogateway";
```

The entry in the services file would be as follows:

```
infogateway 1453/tcp
```

Select a TCP/IP port number (for example, *1453*) for the service name that is unique and available on your system.

23

# Running Problem Service

Problem Service can be started after installation and configuration is complete, and HLAPI has been invoked successfully.

## Starting Problem Service

For AIX and Solaris, Problem Service is started by running the **gw_nxd** executable file. This file is in the **$INST_DIR/$INTERP/InfoMgt/InfoGateway** directory. Other daemon sessions are automatically started by the base process according to what was specified for the session statements in the Problem Service configuration file.

You can start Windows NT system services by opening the control panel folder and double-clicking on the services icon. The name will be displayed in the services list as `TSD390 Problem Service Gateway`. Click on this name, then click on the start button.

You must start Problem Service from an administrator user ID in the Tivoli Management Region (TMR) where Problem Service is installed to use reverse assignment and monitor operations.

## Stopping Problem Service

For AIX and Solaris, use the **shutdown** Problem Service function or locate the process ID of the main Problem Service process and use the **kill** command to stop the daemon. The main process will stop the subprocesses. For more information on the shutdown function, refer to "Shutdown" on page 303.

You can stop Windows NT system services by opening the control panel folder and double-clicking on the services icon. The name will be displayed in the services list as `TSD390 Problem Service Gateway`. Click on this name, then click on the stop button.

## Logging with Problem Service

Problem Service has the ability to write error, trace, and informational messages to a log file. There are several characteristics of the logging function that can be customized. A list of these, along with their default values, are:

**Log name**
: This is the name of the log file. The default log file name used is infogw.log and is created in the current directory.

**Log size**
: This is the maximum size of the log file. The default log size is set to 250000 bytes. Upon reaching this size the current active log is archived and a new log file is

started. One archived backup log is maintained and uses the name of the log file with .bak concatenated to the end. If the default log name is used, the backup file would have the name infogw.log.bak.

Archiving takes place by removing the existing backup file and then renaming the current active log file to the backup name. The new log file is then created.

**Log state**
This is the state of the logger (ON or OFF). The logger is ON by default.

**Log level**
This is the level of logging to be performed. The logger has three levels of logging:

**Level 1**
Only messages listed as errors are written to the log file

**Level 2**
Error messages plus trace messages are written to the log file

**Level 3**
Error, trace, and informational messages are written to the log file

The log level is set to level 3 by default, which causes all messages to be logged.

The logging characteristics can be customized by setting and exporting the following environment variables:

**GWLOGNAME**
Specifies the name of the log file.

**GWLOGSIZE**
Specifies the maximum size of the log file in bytes. The smallest value that can be specified is 5000 bytes. A smaller value results in 5000 bytes being used as the maximum size.

**GWLOGSTATE**
Specifies the state of the logger. The logger is disabled if the value of this variable is set to OFF. Other values cause the logger to be ON.

**GWLOGLEVEL**
Sets the level of logging. The valid values are 1, 2, or 3. A value less than 1 causes 1 to be used as the level and a value more than 3 causes 3 to be used as the level.

The environment variables must be exported from a parent session of the one used by Problem Service.

Each output to the log file will contain the following information:

```
Date     Time     PID   MessageID     MessageText
```

An example output to the log is:

```
01/10/97 16:32:35 18454 APAGM034I Process configuration file blymgc.cfg.
01/10/97 16:32:38 3610  APAGT003I Gateway Reverse Assignment process started
01/10/97 16:32:38 3610  APAGT028I Connect to Tivoli Information Management for z/OS
```

This log file, along with the HLAPI client log and the HLAPI client probe log, can be useful in identifying and locating problems. Information about the two HLAPI client log files can be found in the *Tivoli Information Management for z/OS Client Installation and User's Guide*.

24

# Problem Service Application Programming Information

This chapter is for the programmer who is developing an application that will invoke the Problem Service operations. Refer to the Tivoli Application Development Environment (Tivoli ADE) publications for more information about developing an application in the Tivoli environment.

## Copying the Samples and Files

Copy the samples and files you need to compile calls to the APIs.

- Create a test directory (example: **/home/userid/tivtest**).
- Change directory to the new test directory.
- Copy the files into the new test directory. The files are in **$INST_DIR/../include/$INTERP/InfoMgt/InfoGateway** unless you specified a different directory for header files during the installation of Problem Service.

## Compiling and Link Editing Your Code

Compile and link edit your code with the Problem Service APIs.

- Change to your test directory.
- Edit the Makefile for your setup:
  - Change **TOP** (directory where Tivoli is installed)
  - Change **HERE** (your test directory)
  - For Windows NT only, change **TOOLROOT** (your compiler directory)
  - Change references to tester, tester.c, and tester.o to the appropriate names for your application if you have written your own program that uses the Problem Service operations.
  - Change the compiler commands and libraries to those for your compiler. For AIX, the sample Makefile uses the xlC compiler and the IBMcset libraries. For Windows NT, the sample tester.mak uses the IBM VisualAge® for C++ for Windows® compiler.
- For AIX workstations, enter `ln -s /home/userid/tivtest tivoli` to set up a symbolic link used by the make program to find the files you copied.
- Compile and link edit your program. The sample C program, tester.c, uses the gnu make (gmake) program. For AIX workstations, enter `gmake test` to compile and link the tester.c program using the gnu make program. For Windows NT workstations, enter `make -f tester.mak tester.exe` to compile and link the tester.c program using the gnu make program.

Refer to the Tivoli Application Development Environment publications for more information about creating an application in the Tivoli environment.

## Interface Definition Language Data Types

The following are the interface definition language (IDL) data types used by the Problem Service API:

**string** Null terminated character string

**unsigned long**
32 bit integer

**sequence**
A one dimensional array of elements. The Common Object Request Broker Architecture (CORBA) specification defines the sequence data type for operations that accept or return a set of data structures. Tivoli provides a library of functions for manipulating sequences.

This is an example of a sequence that can be used in coding Problem Service operations:

```
struct GWAttr {
  string name;    // name of data field
  string value;   // value of data field
};

typedef sequence <GWAttr>GWAttrList;
```

The GWAttrList sequence contains a list of elements that represent a record. Each element has a name and value. The name identifies the data field name and the value contains the value associated with the data field.

## Coding Examples for Problem Service Operations

The following code fragments demonstrate how to code the Problem Service operations in an application program. For more detail and to see each example in context, refer to the Bash shell script examples in the **sample** file and the C code examples in the **tester.c** file that are shipped with Problem Service.

### Checkin

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
idlcall $OID InfoGW::checkin \"00000311\"
```

C example:

```
 /*****************************************************/
  /* Variables                                         */
  /*****************************************************/
  Environment     ev;
  Object          oid = OBJECT_NIL;

  /*****************************************************/
  /* Get the object id                                 */
  /*****************************************************/
  oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
  /*****************************************************/
```

```
/* Perform the transaction                          */
/****************************************************/
t_InfoGW_checkin(oid, &ev, Trans_none, "00000311");
```

## Checkout

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
idlcall $OID InfoGW::checkout \"00000311\"
```

C example:

```
/****************************************************/
 /* Variables                                        */
 /****************************************************/
 Environment     ev;
 Object          oid = OBJECT_NIL;

 /****************************************************/
 /* Get the object id                                */
 /****************************************************/
 oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
 /****************************************************/
 /* Perform the transaction                          */
 /****************************************************/
 t_InfoGW_checkout(oid, &ev, Trans_none, "00000311");
```

## Delete

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
idlcall $OID InfoGW::delete \"00000311\"
```

C example:

```
/****************************************************/
 /* Variables                                        */
 /****************************************************/
 Environment     ev;
 Object          oid = OBJECT_NIL;

 /****************************************************/
 /* Get the object id                                */
 /****************************************************/
 oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
 /****************************************************/
 /* Perform transaction                              */
 /****************************************************/
 t_InfoGW_delete(oid, &ev, Trans_none, "00000311");
```

## Ping

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW
PINGRES=`idlcall $OID InfoGW::ping`
```

C example:

```
 /****************************************************/
 /* Variables                                        */
 /****************************************************/
 Environment     ev;
 Object          oid = OBJECT_NIL;
 PingResult      pr;
```

```
/****************************************************/
/* Get the object id                                */
/****************************************************/
oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
/****************************************************/
/* Perform transaction                              */
/****************************************************/
pr = t_InfoGW_ping(oid, &ev, Trans_none);
```

## Propagate

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
RNID1=`idlcall $OID InfoGW::propagate { 2 \
     {\"Originator\"\"Smith,Bill\"}\
     {\"Status\"\"Pending\"}} \"Helpdeskapp\"`
```

C example:

```
/****************************************************/
/* Variables                                        */
/****************************************************/
Environment  ev;
Object       oid = OBJECT_NIL;
GWAttrList   gwattrlist;
GWAttr       tmpattr;
char         * rnid;

/****************************************************/
/* Get the object id                                */
/****************************************************/
oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
/****************************************************/
/* Initialize the sequence of name value pairs      */
/****************************************************/
seq_init((sequence_t * ) &gwattrlist );
/*******************************************************/
/* Set the element values and append them to the sequence */
/*******************************************************/
tmpattr.name  = ml_ex_strdup("Originator");
tmpattr.value = ml_ex_strdup("Smith,Bill");
seq_add( (sequence_t *)&gwattrlist,&tmpattr,sizeof(GWAttr) );
tmpattr.name  = ml_ex_strdup("Status");
&tmpattr.value = ml_ex_strdup("Pending");
seq_add( (sequence_t *)&gwattrlist,&tmpattr,sizeof(GWAttr) );
/********************************************************/
/* Perform the transaction and save the record id         */
/********************************************************/
rnid=t_InfoGW_propagate(oid,&ev,Trans_none,&gwattrlist,"Helpdeskapp");
/********************************************************/
/* Free the input sequence's buffer                       */
/********************************************************/
seq_free_buffer((sequence_t *) &gwattrlist);
```

## Retrieve

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
RESULT=`idlcall $OID InfoGW::retrieve \"00000311\"\"Helpdeskapp\"`
```

C example:

```
/****************************************************/
/* Variables                                        */
/****************************************************/
Environment    ev;
```

```
  Object        oid = OBJECT_NIL;
  GWAttrList    myrec;

  /****************************************************/
  /* Get the object id                                */
  /****************************************************/
  oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
  /****************************************************************/
  /* Perform the transaction                                      */
  /****************************************************************/
  myrec=t_InfoGW_retrieve(oid,&ev,Trans_none,"00000311","Helpdeskapp");
```

## Search

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
RESULT2=`idlcall $OID InfoGW::search { 2 \
     {\"Originator\"\"Smith,Bill\"}
     {\"Status\"\"Pending\"}} \"Helpdeskapp\"`
```

C example:

```
  /****************************************************/
  /* Variables                                        */
  /****************************************************/
  Environment       ev;
  Object            oid = OBJECT_NIL;
  GWAttrList        gwattrlist;
  GWAttr            tmpattr;
  SearchResultList  mysearchlist;

  /****************************************************/
  /* Get the object id                                */
  /****************************************************/
  oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
  /************************************************************/
  /* Initialize the sequence of name value pairs              */
  /************************************************************/
  seq_init((sequence_t * ) &gwattrlist);
  /************************************************************/
  /* Set the element values and append them to the sequence   */
  /************************************************************/
  tmpattr.name  = ml_ex_strdup("Originator");
  tmpattr.value = ml_ex_strdup("Smith,Bill");
  seq_add( (sequence_t *)&gwattrlist, &tmpattr, sizeof(GWAttr) );
  tmpattr.name  = ml_ex_strdup("Status");
  tmpattr.value = ml_ex_strdup("Pending");
  seq_add( (sequence_t *)&gwattrlist, &tmpattr, sizeof(GWAttr) );
  /************************************************************/
  /* Perform the transaction                                  */
  /************************************************************/
  mysearchlist = t_InfoGW_search(oid, &ev, Trans_none,
                               &gwattrlist, "Helpdeskapp");
  /************************************************************/
  /* Free the input sequence's buffer                         */
  /************************************************************/
  seq_free_buffer((sequence_t *) &gwattrlist);
```

## Shutdown

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
idlcall $OID InfoGW::shutdown
```

C example:

```
/****************************************************/
/* Variables                                        */
/****************************************************/
Environment       ev;
Object            oid = OBJECT_NIL;

/****************************************************/
/* Get the object id                                */
/****************************************************/
oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
/****************************************************/
/* Perform transaction                              */
/****************************************************/
t_InfoGW_shutdown(oid, &ev, Trans_none);
```

## Transfer

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
RNID=`idlcall $OID InfoGW::transfer { 2 \
    {\"Originator\"\"Smith,Bill\"} \
    {\"Status\"\"Pending\"}} \Helpdeskapp\"`
```

C example:

```
/****************************************************/
/* Variables                                        */
/****************************************************/
Environment  ev;
Object       oid = OBJECT_NIL;
GWAttrList   gwattrlist;
GWAttr       tmpattr;
char         * rnid;

/****************************************************/
/* Get the object id                                */
/****************************************************/
oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
/*****************************************************************/
/* Initialize the sequence of name value pairs                  */
/*****************************************************************/
seq_init((sequence_t *)  &gwattrlist);
/*****************************************************************/
/* Set the element values and append them to the sequence.      */
/*****************************************************************/
tmpattr.name  = ml_ex_strdup("Originator");
tmpattr.value = ml_ex_strdup("Smith,Bill");
seq_add( (sequence_t *) &gwattrlist, &tmpattr, sizeof(GWAttr) );
tmpattr.name  = ml_ex_strdup("Status");
tmpattr.value = ml_ex_strdup("Pending");
seq_add( (sequence_t *) &gwattrlist, &tmpattr, sizeof(GWAttr) );
/*****************************************************************/
/* Perform the transaction and save the record id               */
/*****************************************************************/
rnid=t_InfoGW_transfer(oid,&ev,Trans_none, &gwattrlist,"Helpdeskapp");
/*****************************************************************/
/* Free the input sequence's buffer                              */
/*****************************************************************/
seq_free_buffer((sequence_t *)  &gwattrlist);
```

## Update

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
idlcall $OID InfoGW::update { 2 \
     {\"Status\"\"Closed\"}
     {\"rnid\"\"311\"}} \"Helpdeskapp\"
```

C example:

```
/*****************************************************/
/* Variables                                         */
/*****************************************************/
Environment  ev;
Object       oid = OBJECT_NIL;
GWAttrList   gwattrlist;
GWAttr       tmpattr;

/*****************************************************/
/* Get the object id                                 */
/*****************************************************/
oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
/****************************************************************/
/* Initialize the sequence of name value pairs                 */
/****************************************************************/
seq_init((sequence_t * )  &gwattrlist);
/****************************************************************/
/* Set the element values and append them to the sequence.     */
/****************************************************************/
tmpattr.name  = ml_ex_strdup("Status");
tmpattr.value = ml_ex_strdup("Closed");
seq_add( (sequence_t *) &gwattrlist,&tmpattr,sizeof(GWAttr) );
tmpattr.name  = ml_ex_strdup("rnid");
tmpattr.value = ml_ex_strdup("00000311");
seq_add( (sequence_t *) &gwattrlist,&tmpattr,sizeof(GWAttr) );
/****************************************************************/
/* Perform the transaction                                      */
/****************************************************************/
t_InfoGW_update(oid, &ev, Trans_none,  &gwattrlist, "Helpdeskapp");
/****************************************************************/
/* Free the input sequence's buffer                             */
/****************************************************************/
seq_free_buffer((sequence_t *) &gwattrlist);
```

## Tivoli Application Development Environment (ADE) Exceptions

Tivoli ADE exceptions are used to return error information to your application. Problem Service defines the ExInfoGateway exception. Refer to the Tivoli Application Development Environment documentation for more information on these exceptions.

### ExInfoGateway Exception

The following data is inherited from ExException:

**string type_name**
Name of exception type

**string catalog**
Name of message catalog

**long key**
Message catalog key

**string default_message**
Default message, when the message catalog is unavailable

**long stamp**
Date stamp

**Msgcontext Msg_context**
Context of exception raised

The following are ExInfoGateway data:

**string message**
: Name of C++ exception

**HICAReturnCode**
: Tivoli Information Management for z/OS API return code

**HICAReasonCode**
: Tivoli Information Management for z/OS API reason code

## Examples of Gateway Exceptions

```
Try {
  t_InfoGW_checkout(oid, &ev, Trans_none,recordID);
}
Catch(ExInfoGateway,ex){//catches Gateway exceptions and
                          //any exceptions derived from
                          //ExInfoGateway
}
Catch(ExException,ex){  //catches ExExceptions and any
                          //exceptions derived from ExException
}
CatchAll() {            //catches all exceptions
}
```

# 25 Customizing User Exit Routines for the Problem Service Daemon

Problem Service assumes that all data is character data. Some data conversions, such as the truncation of data, are supported. Problem Service does not know the Tivoli Information Management for z/OS database record structure, and the Tivoli Information Management for z/OS HLAPI does not automatically convert data, so you must define how truncations and other conversions are to be performed.

Some types of mapping syntax enable you to define fields and rules for mappings for each type of transaction. The mapping rule for a particular field can be different for different transactions.

## Supported Data Conversions

Here is a list of the mapping conversions that are supported, followed by a description of each listed item:

- Truncation
- Convert one character to another character
- Convert specific field value to another value
- Date/time conversion
- Freeform text
- Default data
- Field combining (concatenation)
- Substring and sub-word
- Exit routines

### Truncation

Data can be truncated to a specified length. For example, assignee in an application’s record is 90 characters, while assignee in a Tivoli Information Management for z/OS database record is only 15 characters, so you can choose to truncate after the first 15 characters.

### Convert One Character to Another Character

An application’s field could contain blanks and commas while the corresponding Tivoli Information Management for z/OS field might allow only one word and not allow commas. The blanks and commas in the field are converted to a specified character that you can choose.

The Tivoli Information Management for z/OS record field may allow multiple words (for example, first and last name). In this case, the blanks and commas would be converted to a specified separator character. This allows the HLAPI client to indicate that the data contains multiple words (separator character separates each word).

## Convert Specific Field Value to Another Value

An application's field might allow hardcoded values that are different from the corresponding Tivoli Information Management for z/OS problem fields. Each value that does not match is converted to a specified value.

## Date/Time Conversion

An application's time stamp can be a combination of date and time. The date part and the time part are put into the Tivoli Information Management for z/OS record's respective date field and time field. Tivoli Information Management for z/OS enables you to choose which external date and time formats to use (exit routine is used to convert from internal to external and vice versa).

The HLAPI client accepts dates and times in external format (this could be different for each Tivoli Information Management for z/OS site). Problem Service allows the specification of a C exit routine to convert the application's record date/time into dates and times to be given to Tivoli Information Management for z/OS.

**Note:** This means that a C compiler is a prerequisite if you want to use your own user exits.

The date and time conversion exit can manipulate the date and time to support time zone differences between Tivoli Information Management for z/OS and your applications.

## Freeform Text

An application's freeform textual field is converted to Tivoli Information Management for z/OS freeform text. An application's text data might be just a stream of characters with no indications of new lines. In this case, specify the length of the corresponding Tivoli Information Management for z/OS text line so that your application's text can be split into Tivoli Information Management for z/OS text lines. When returning freeform text, the text lines can be converted to a data stream before being given to your application.

## Default Data

You can specify hardcoded data for a field, either on an unconditional basis or only if the source data field is empty.

## Field Combining (Concatenation)

You can combine multiple fields into one target field. For example, several of your application's fields might map into just one Tivoli Information Management for z/OS field.

## Substring and Sub-Word

You can choose to map only a part of a field into the target field.

## Exit Routines

An exit routine can be specified to perform whatever conversions you choose. This exit routine must be written in the C programming language.

## Specifying User Exits for Conversions

You can specify exit routines to perform data conversions. These can be routines you write yourself or those provided by Problem Service.

If you want to write your own exit routine, code it as shown in the following example:

```
char* main(int argumentCount, const char ** argumentArrayPointer);
```

To ensure that control returns to the mapping facility after the invocation of a user exit, it must be linked with the entry point as `main` and not a compiler-generated routine. Refer to the link options of the compiler you are using for instructions.

Copy the newly created user exit routine to the directory where the gw_nxd daemon executable exists, so that Problem Service can use it.

Several user exits to perform data conversions are provided. The first argument for all of the exits is *data*, and is either absent, a field name, a literal string, or another subroutine specification. This is also true for any other argument requiring a string.

Table 41 is a list of the supplied user exits. Each user exit with examples of use is described following the table.

*Table 41. Supplied User Exits*

| User Exit | Description |
|---|---|
| change | Returns specified string changes. |
| fromIMDate | Converts a date format year value. |
| fromIMPriority | Maps priority values. |
| fromIMTime | Converts a military time format. |
| nullDefault | Returns a specified value when the target is null. |
| stripLeading | Strips leading characters. |
| subString | Returns a specified substring. |
| toIMDate | Converts a date format year value. |
| toIMTime | Converts a time format to military time format. |
| translate | Returns a specified character translation. |
| translateWord | Returns a specified word translation. |
| words | Returns a specified substring. |

### change

This user exit returns specified string changes of the target *data*. The format is:

```
change(data, sourceWord1, targetWord1,
             sourceWord2, targetWord2, ...)
```

The string given by *sourceWord1*, where found within *data*, is changed to the string given by *targetWord1*. It then changes the string given by *sourceWord2*, where found in the result of the first operation, to the string given by *targetWord2*. This continues until all source strings have been processed. If the last matching target string is missing, it defaults to null.

For example, if *fieldName* is *abcdefghijklm* then:

```
change(fieldName,"abc","def","def","ghi") returns "ghighighijklm"
change("abcdefghijklm","def")            returns "abcghijklm"
```

## fromIMDate

This user exit converts a *yy* date format to a *yyyy* date format. The format is:

```
fromIMDate(IMDate)
```

A date in the format `mm/dd/yy` is converted to `mm/dd/yyyy`, where year characters 50 through 99 represent the years 1950 through 1999 and year characters 00 through 49 represent the years 2000 through 2049.

For example,

```
fromIMDate("12/05/49") returns "12/05/2049"
```

## fromIMPriority

This user exit maps priority values. The format is:

```
fromIMPriority(IMPriority)
```

It maps values 6 through 99 to 5, while not altering values 0 through 5.

For example,

```
fromIMPriority("21") returns "5"
fromIMPriority("0")  returns "0"
```

## fromIMTime

This user exit converts a military time format. The format is:

```
fromIMTime(IMTime)
```

The military time format of `hh:mm` is converted to a time in the format `hh:mm:ss am` or `hh:mm:ss pm`, by adding a seconds field of 00 and am or pm.

For example,

```
fromIMTime("13:34") returns "01:34:00 pm"
```

## nullDefault

This user exit returns a specified value when the target is null. The format is:

```
nullDefault(data, defaultValue)
```

The value *defaultValue* is returned when *data* is null. Otherwise, the value for *data* is returned.

For example, if *fieldName* is *abc* then:

```
nullDefault(fieldName,"default") returns "abc"
nullDefault(,fieldName)          returns "abc"
nullDefault(,)                   returns ""
```

## stripLeading

This user exit strips the leading characters from the target *data*. The format is:

```
stripLeading(data, stripCharacters)
```

For example, if *fieldName* is *0000200* then:

```
stripLeading(fieldName,"0")       returns "200"
stripLeading(fieldName)           returns "0000200"
stripLeading("000000","0")        returns ""
stripLeading("wordwordzz","word") returns "zz"
```

## subString

This user exit returns a specified substring of the target *data*. The formats are:

```
subString(data, startPosition)
subString(data, startPosition, length)
subString(data, startPosition, length, padCharacter)
```

**Where:**

**startPosition**
Starting index position of the substring. If the index is beyond the end of the string, the function returns a null string.

**length** The length of the substring. If the substring extends beyond the end of the string, the substring is padded with the character given by the *padCharacter* argument. If length is not specified, the substring goes from the starting position to the end of the string.

**padCharacter**
The character to use as padding if the substring extends beyond the end of the string. The default pad character is a single space.

For example, if *fieldName* is *abcdef* then:

```
substr(fieldName,2,3)           returns "bcd"
substr(fieldName,4,5)           returns "def  "
substr(substr("abcdef",2,3),2,1) returns "c"
substr("abcdef",7,1)            returns " "
```

## toIMDate

This user exit converts a *yyyy* date format to a *yy* date format. The format is:

```
toIMDate(foreignDate)
```

A date in the format `mm/dd/yyyy` is converted to `mm/dd/yy`, where year characters 50 through 99 represent the years 1950 through 1999 and year characters 00 through 49 represent the years 2000 through 2049.

For example,

```
toIMDate("01/31/1950") returns "01/31/50"
```

## toIMTime

This user exit converts a time format to military time format. The format is:

```
toIMTime(foreignTime)
```

A time in the format `hh:mm:ss am` or `hh:mm:ss pm` is changed to military time `hh:mm`, by dropping the seconds.

For example,

```
toIMTime("12:34:56 am") returns "00:34"
```

## translate

This user exit returns a specified character translation of the target *data*. The format is:

```
translate(data, inputCharacters, outputCharacters)
```

The characters given by *inputCharacters*, where found within *data*, are changed to the characters given by *outputCharacters*. If not specified, *outputCharacters* defaults to spaces.

For example, if *fieldName* is *a,b,c,d* then:

```
translate(fieldName,","," ")    returns "a b c d"
translate("a,b,c,d",fieldName)  returns "       "
```

## translateWord

This user exit returns a specified word translation of the target *data*. The format is:

```
translateWord(data, sourceWord1, targetWord1,
                    sourceWord2, targetWord2, ...)
```

The string given by *sourceWordN* is changed to the string given by *targetWordN*, if found within *data*. If the last matching target is missing, it defaults to null. If *sourceWord1* is *, any result for *data* is a match.

For example, if *fieldName* is *word* then:

```
translateWord(fieldName,fieldName,"bird") returns "bird"
translateWord("word","word")              returns ""
```

## words

This user exit returns a specified substring of the target *data*. The format is:

```
words(data, firstWord)
words(data, firstWord, numberOfWords)
```

It begins with the word in the word index position given by *firstWord* (words are separated by spaces). If the index given by *firstWord* is not valid, the function returns a null string. The *numberOfWords* argument can be used to specify how many words to include in the substring. If the *numberOfWords* argument is not specified, all the words to the end of the string are included in the substring. The substring contains all the word separators (spaces) that are included in the original string.

For example, if *fieldName* is *a b c d* then:

```
words(fieldName,2,1)               returns "b"
words(subString(fieldName,3,3),2,2) returns "c"
words("a b c d",5,1)               returns ""
```

# IV — Tivoli Service Desk Bridge

26

# Tivoli Service Desk Bridge Overview

**Note:** Tivoli Service Desk (TSD) is a suite of applications; Tivoli Problem Management is one of those applications. In the following chapters on Tivoli Service Desk Bridge, TSD represents Tivoli Problem Management; Information Management represents Tivoli Information Management for z/OS.

Tivoli Service Desk (TSD) is a network help desk system that enables help desk analysts to register calls and resolve problems. With TSD, help desk analysts can simultaneously access a large database of problems and solutions, track customer calls and problems, and transfer calls and problems that your help desk handles. TSD has traditionally been a tool that is used in a workstation environment. Tivoli Information Management for z/OS provides functions similar to that of TSD, but traditionally has been a tool that is used in a host environment.

Tivoli Information Management for z/OS can exchange problem records with TSD installed on a Windows NT workstation platform. Working from TSD, a help desk analyst can request that a problem record be transferred from a TSD help desk analyst to an Information Management help desk analyst, and, working from Information Management, the reverse is true.

Whether a problem record resides in the TSD database or in the Information Management database, a help desk analyst who has knowledge of a problem record (usually by having worked on the record previously) can request transfer of that record. When transferred, information about the problem can be added, and the record can be transferred to another analyst if that is the appropriate disposition of the record.

TSD maintains a database of problem records that is separate from the Information Management database of problem records. A problem record can reside in both databases at the same time, but it is “owned” by either Information Management or TSD. In Information Management, this “ownership” designation is determined by the **Person role** contained in the *people record* of the user to whom the problem is assigned. Each user who will access a problem record is designated as an Information Management user known only to Information Management (person role=TSD390), a TSD user or group known to Information Management (person role=TSDUSER or TSDGROUP), or a Information Management user known to TSD (role=TSD390&TSD). If a problem record is assigned to a user whose role is TSDUSER or TSDGROUP, the record is owned by TSD. If a problem record is assigned to a user whose role is anything other than TSDUSER or TSDGROUP, the record is owned by Information Management.

A people record must be created for each user who can be assigned to problem records. A people record created for an Information Management user will contain the **Person role** of either TSD390&TSD or TSD390. A people record can also be created for a TSD user and will contain a **Person role** of TSDUSER or TSDGROUP. The creation of people records is described in the *Tivoli Information Management for z/OS Program Administration Guide and Reference*.

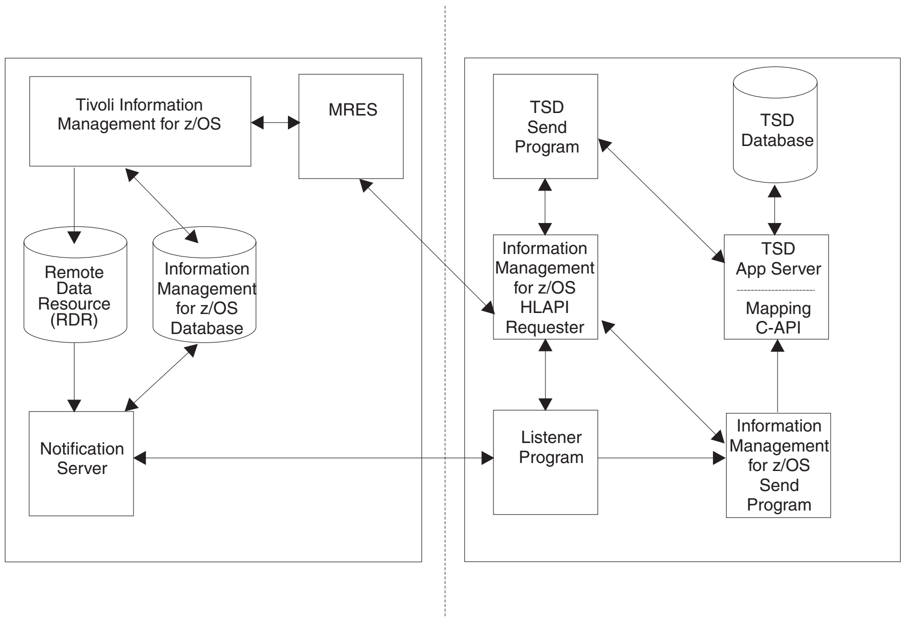

Tivoli Information Management for z/OS
MRES
Remote Data Resource (RDR)
Information Management for z/OS Database
Notification Server
TSD Send Program
TSD Database
Information Management for z/OS HLAPI Requester
TSD App Server
Mapping C-API
Listener Program
Information Management for z/OS Send Program

*Figure 16. TSD Bridge Overview*

The diagram in Figure 16 represents the interaction between Tivoli Information Management for z/OS and TSD. The left portion of the diagram represents activity on the Information Management side; the right portion of the diagram represents activity on the TSD side. Each problem management application, Information Management and TSD, has its own database.

## Problem Records and People Records

Information Management notifies TSD when Information Management records meet certain criteria. Information Management notifies TSD when:

- A *problem record* is assigned to a user whose **Person role** is either TSDUSER or TSDGROUP. Refer to the *Tivoli Service Desk Networking Guide* for more information about these **Person role**s.
- A *problem record* for which a command of *Resume Ownership*, *Refresh*, or *Send a Solution* is issued.
- A *people record* with a **Person role** of TSD390&TSD is created or updated.

When a problem record or a people record meets any of these criteria, Information Management stores notification data in the record and, if the remote data resource (RDR) is open, places the notification data on that RDR (general information about RDRs is contained in the *Tivoli Information Management for z/OS Program Administration Guide and Reference*).

## The Notification Server

The notification server manages TSD notification. It is a TSX that is invoked by starting Information Management as a batch job. When the notification server is started, it opens a control RDR, establishes a long-running TCP/IP connection with the TSD listener program, and opens the data RDR. The notification server then searches the TSD390 database for records that need to processed by TSD and sends the notification data for those records to the TSD listener program. Next, the notification server waits on the data RDR for work. As Information Management places notification data on the RDR, the notification server removes it from the RDR and sends it to the TSD listener program.

If the Notification Server is shut down in the middle of a process, a database administrator must use the TSD Cleanup tool (described in the *Tivoli Information Management for z/OS Program Administration Guide and Reference*) to clear records from Notification Server queue.

## The Listener Program

When the TSD listener program receives data from the notification server, the listener program uses the Information Management HLAPI/NT Requester to retrieve information from the Information Management database. The information is sent to the Information Management Send program, where s-word values are mapped to the TSD fields. The mapped data is written to the TSD database.

27

# Tivoli Service Desk Bridge Setup

Setting up the Tivoli Service Desk Bridge involves setup for Information Management, described in "Information Management Setup", and setup for TSD, described in "TSD Setup" on page 350. It is assumed that you have already installed Tivoli Information Management for z/OS Version 7.1.

## Hardware Requirements

There are no special hardware requirements for the Tivoli Service Desk Bridge.

## Software Requirements

The software requirements for the Tivoli Service Desk Bridge follow:

- Tivoli Service Desk Version 6.0.
- The Tivoli Service Desk Bridge can interface with TSD installed on a Windows NT workstation platform. The TSD 6.0 installation package includes the Tivoli Service Desk for OS/390 1.2 HLAPI/NT Client API. This code will work with Tivoli Information Management for z/OS 7.1.
- TCP/IP connectivity between the notification server and the listener program.

In addition, if you are going to create Tivoli Enterprise Console (TEC) events to handle Tivoli Service Desk Bridge errors, you will need the following:

- Tivoli Information Management for z/OS TEC Event Adapter
- Tivoli Event Integration Facility for OS/390 (5697–C74)
- See "Setting Up Error Processing for the TSD Listener Program" on page 350 for more information about handling errors

### Database Requirements

To use the Tivoli Service Desk Bridge, create your Information Management database with an SDIDS key length of 34. For information about SDIDS key length, refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*.

## Information Management Setup

These are the steps that you must follow for Information Management before using the Tivoli Service Desk Bridge:

1. Set up an MRES with TCP/IP that pre-starts API sessions or an MRES with APPC that pre-starts API sessions. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for information on setting up an MRES and details on pre-started API sessions.

Sample JCL to define an MRES procedure is shipped as BLMMRES in the SBLMSAMP sample library. A sample MRES parameters member is shipped as BLMMRESP in the SBLMSAMP sample library. The following parameters member values are recommended:

- MULTIPLE_RESPONSE_FORMAT=PHRASE
- TABLE_COUNT=10

2. Set up the notification server. The information needed to do this can be found in "Setting Up the Notification Server" on page 321.
3. Load data model records (data view records and data attribute records) for people records and problem records. The procedure for doing this is described in "Data Model Records" on page 321.
4. The Problem Summary Display panel, BLG0S010, is shipped from Tivoli with selection **11. TSD Bridge display**; if you have not modified BLG0S010, use this panel shipped from Tivoli. If you have customized BLG0S010 or you use a different problem summary display panel, update it to include a selection for TSD Bridge display. The procedure for adding a new selection for TSD Bridge display is described in "Updating Panel BLG0S010" on page 323.
5. The Problem Inquiry Summary panel, BLG0E090, is shipped from Tivoli with selection **7. TSD Bridge data**; if you have not modified BLG0E090, use this panel shipped from Tivoli. If you have customized BLG0E090 or you use a different problem inquiry summary panel, update it to add a selection for TSD Bridge data. The procedure for adding a new selection for TSD Bridge data is described in "Updating Panel BLG0E090" on page 331.
6. BLG1A111 is the Problem Record File panel. If you use the panel as shipped from Tivoli, use the Panel Modification Facility (PMF) to copy panel BLG1A11Z from the Tivoli base panel data set and rename it to panel BLG1A111 in your read panel data set; a procedure for doing this is described in "Copying Panel BLG1A11Z" on page 345. If you have modified BLG1A111 or use a different problem record file panel, use PMF to update the panel to invoke the TSX BLGTSDPT; a procedure for doing this is described in "Updating Panel BLG1A111" on page 338.
7. Use the Panel Modification Facility (PMF) to copy panel BLM1B04Z from the Tivoli base panel data set and rename it to panel BLM1B040 in your read panel data set; a procedure for doing this is described in "Copying BLM1B04Z" on page 347.
8. The TSX BLGTSDRQ is located in the SBLMTSX data set. The value specified for the data RDR specified in BLGTSDRQ must be identical to the value specified for the data RDR specified in the notification server, BLGTSDNS. See "Setting Up the Notification Server" on page 321.
9. The Tivoli-supplied TSX BLGTSDPS, described on page 363, is used to send the resolution for a closed problem record to TSD. The default value for ″closed″ is STAC/CLOSED. If you use something other than STAC/CLOSED to indicate that a problem is closed, modify BLGTSDPS, located in the SBLMTSX data set. Copy BLGTSDPS to your TSX data set and update the value for the variable ″closed″ to the p-word and value for your installation.
10. Start the MRES that you defined in step 1 on page 319. Information on how to start the MRES can be found in the *Tivoli Information Management for z/OS Client Installation and User's Guide*.

11. Start the notification server. The procedure for doing this is described in “Starting the Notification Server” on page 349.
12. Verify that all steps listed in “TSD Setup” on page 350 have been completed.
13. Create people records for Information Management users to whom problem records will be assigned by TSD. For information about creating people records, see the *Tivoli Information Management for z/OS Program Administration Guide and Reference*.

## Setting Up the Notification Server

The notification server manages TSD notification. It is a Information Management TSX that is run by a batch job. The notification server source is BLGTSDNS and it is in the SBLMTSX data set. You, as the Information Management systems programmer, should copy BLGTSDNS to your TSX data set and update it according to your environment and setup. At a minimum, update the TSX to specify the IP address and port number of the TSD listener program. Also, review and update the other processing options as appropriate. Processing options are specified by the following variables:

**p_ipaddr**
The IP address, entered in dotted decimal format, of the workstation on which the TSD listener program is running; the value of this variable in the TSX BLGTSDNS as shipped from Tivoli is 000.000.000.000. Change this value to the IP address of the workstation on which the TSD listener program is running.

**p_port#**
The TCP/IP port number of the TSD listener program; the value of this variable in the TSX BLGTSDNS as shipped from Tivoli is 0000. Change this value to the port number of the TSD listener program. Refer to the *Tivoli Service Desk Networking Guide* for more information on the listener program.

**p_maxwait**
The maximum time in seconds to try to connect to the TSD listener program. When the time specified expires, processing is terminated; the value of this variable in the TSX BLGTSDNS as shipped from Tivoli is 86400.

**p_waitsec**
The number of seconds to wait before retrying a particular function; the value of this variable in the TSX BLGTSDNS as shipped from Tivoli is 5.

**p_cntlrdr**
The name of the control RDR monitored by the TSX BLGTSDNS; the value of this variable in BLGTSDNS as shipped from Tivoli is BLGTSDC1.

**p_datardr**
The name of the data RDR monitored by the TSX BLGTSDNS; the value of this variable in BLGTSDNS as shipped from Tivoli is BLGTSDD1. If you change this name, change the value of `p_datardr` in the Tivoli-supplied TSX BLGTSDRQ to match. The TSX BLGTSDRQ is located in the SBLMTSX data set.

## Data Model Records

In order to use the Tivoli Service Desk Bridge, load the data model records described in “Loading Data Model Records” on page 322. If you have created data model records for your problem and change records, you must modify them in the manner described in “Customizing Data Model Records” on page 322.

## Loading Data Model Records

Load the Tivoli Information Management for z/OS data model records from the SBLMRCDS data set library. Use batch job BLHRCDSJ in the SBLMSAMP sample library to load these records. The *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* contains additional information about loading data model records.

The BLHRCDSJ job unflattens the data model records and adds them to the Tivoli Information Management for z/OS read/write database specified by the SESS parameter. Before you can run BLHRCDSJ, edit the JCL, using the instructions in the JCL comments.

These are the list members used to load the data model records required for the Tivoli Service Desk Bridge:

**List Members**

**BLHLRBAS**
The list containing the people record attributes. If you can display record BLH&DATE, then the records in BLHLRBAS have already been loaded into your database. You will need to load the records in this list if they have not already been loaded.

**BLMLRDSK**
The list containing the problem record attributes. If you can display record BLM&REQN, then the records in BLMLRDSK have already been loaded. You will need to load the records in this list if they have not already been loaded.

**BLHLRBRG**
The list containing other attributes and views to enable the Tivoli Service Desk Bridge support. You must load the records in the BLHLRBRG list.

## Customizing Data Model Records

The BLHPROB and BLHCHANG data views are provided as sample data views for problem and change records. Use these data views as is or as models if you choose to create views for problem and change records. If you have created a problem record data view that defines your problem application, you can use this with the Tivoli Service Desk Bridge only if you add the Tivoli-supplied data attribute records listed below to your problem record data view. Likewise, if you have created a change record data view that defines your change application, use this with the Tivoli Service Desk Bridge only if you add the Tivoli-supplied data attribute records listed below to your change record data view. If you do not have a problem record data view (and the associated data attributes for your problem application) or a change record data view (and the associated data attributes for your change application), use the TSX BLGTDMBL to build the required data attributes and views. See the *Tivoli Information Management for z/OS Panel Modification Facility Guide* for more information about running BLGTDMBL.

### Data Attribute Records

In order to use the Tivoli Service Desk Bridge, add the following data attribute records to your problem record data view (these data attribute records are in addition to BLH&DATE, BLH&TIME, BLH&CLAE, BLH&DATM, BLH&TIMM, and BLH&USER, which should already be part of your problem record data view):

**BLH&NFID**
Notify User ID

**BLH&OSTE**
Owning Site ID

**BLH&PMID**
TSD Record ID

**BLH&RDAT**
Date Last Refreshed

**BLH&RTIM**
Time Last Refreshed

**BLM&002C**
Management Application Entry S-word (required for Create and Inquiry)

**BLH&14FF**
TSD Bridge Flag

**BLM&URN0**
Problem ID

### Change Record Data View

In order to use the Tivoli Service Desk Bridge, add the following data attribute records to your change record data view (these data attribute records are in addition to BLH&DATE, BLH&TIME, BLH&CLAE, BLH&DATM, BLH&TIMM, and BLH&USER, which should already be part of your change record data view):

**BLH&PMID**
TSD Record ID

**BLM&002C**
Management Application Entry S-word (required for Create and Inquiry)

### Other Supplied Data View Records

The following data view records are supplied for use with the Tivoli Service Desk Bridge and should not be changed:

**BLHPEOPL**
Tivoli Service Desk Bridge People Record View

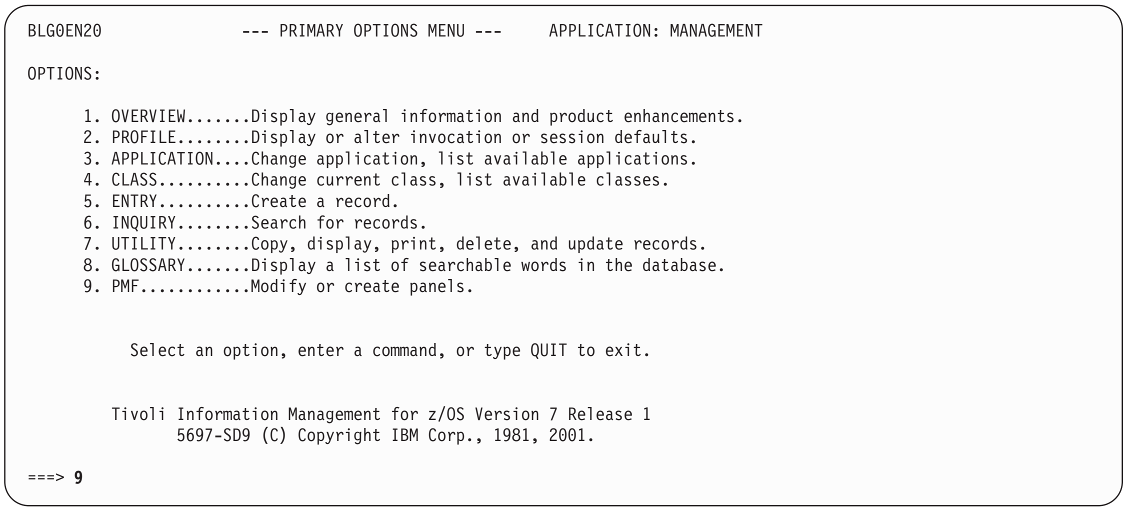
**BLHPLRPY**
Tivoli Service Desk Bridge People Request Reply View

**BLHPPRPY**
Tivoli Service Desk Bridge Problem Request Reply View

## Updating Panel BLG0S010

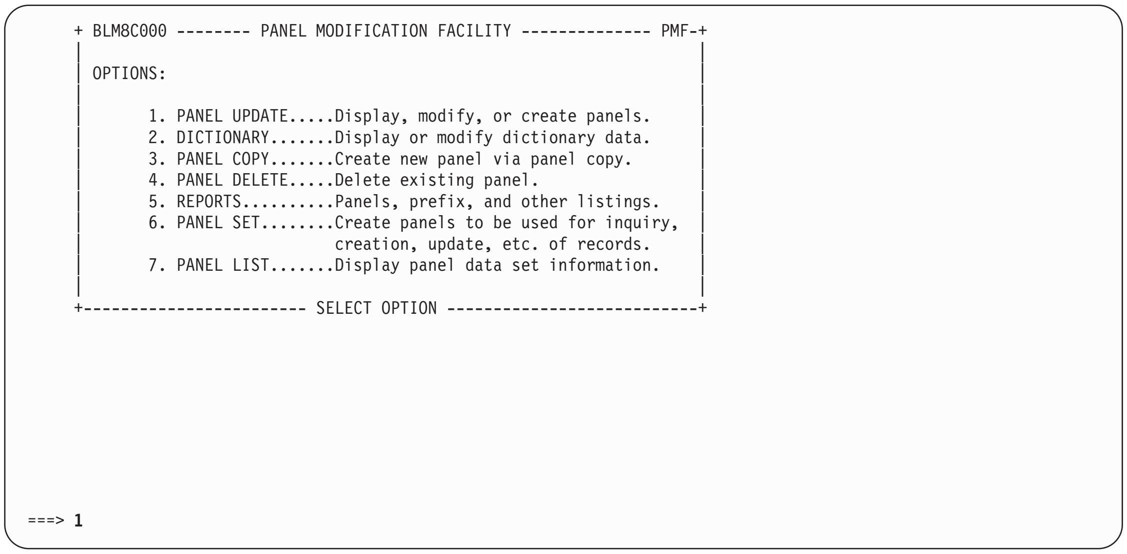
The Tivoli-supplied BLG0S010 contains **11. TSD Bridge display.** as a selection. If you use the Tivoli-supplied BLG0S010, you will not need to perform this modification. However, if you have customized BLG0S010 or you use a different Problem Summary Display panel, you will need to update your modified panel to include a selection for using the Tivoli Service Desk Bridge. This selection will go to the Tivoli Service Desk Bridge Display panel BLG0L700. Panel BLG0L700 provides methods that enable you to resume ownership of a problem record that was transferred to TSD, refresh a problem record that was transferred to TSD with current TSD information, or send the solution for a problem record to TSD. Following are the steps to add that selection.

From BLG0EN20, the Primary Options Menu, type **9** and press Enter to begin the PMF process.



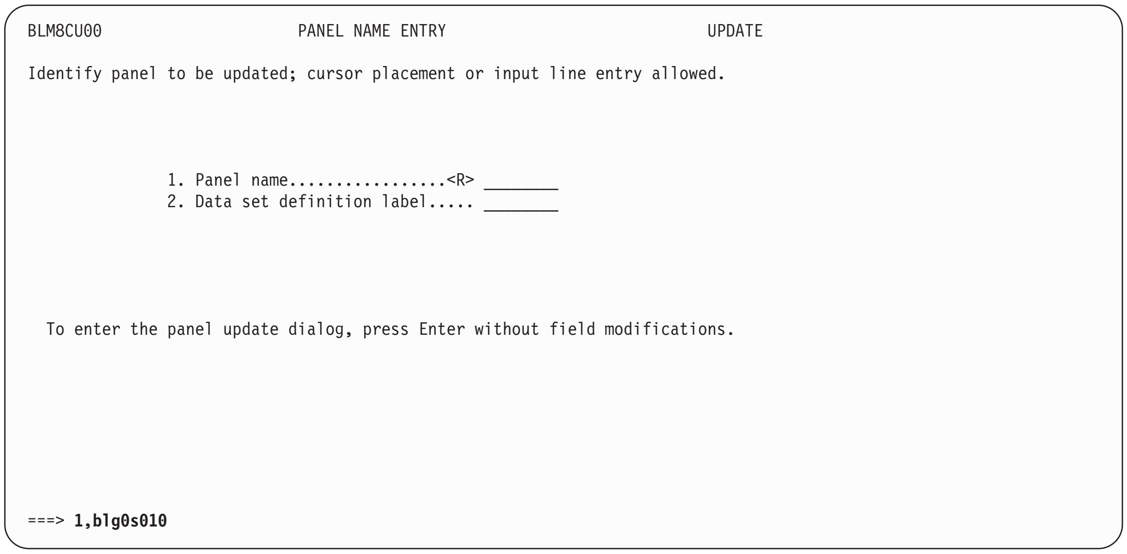
```
BLG0EN20              --- PRIMARY OPTIONS MENU ---     APPLICATION: MANAGEMENT

OPTIONS:

      1. OVERVIEW.......Display general information and product enhancements.
      2. PROFILE........Display or alter invocation or session defaults.
      3. APPLICATION....Change application, list available applications.
      4. CLASS..........Change current class, list available classes.
      5. ENTRY..........Create a record.
      6. INQUIRY........Search for records.
      7. UTILITY........Copy, display, print, delete, and update records.
      8. GLOSSARY.......Display a list of searchable words in the database.
      9. PMF............Modify or create panels.


           Select an option, enter a command, or type QUIT to exit.


         Tivoli Information Management for z/OS Version 7 Release 1
                5697-SD9 (C) Copyright IBM Corp., 1981, 2001.

===> 9
```

Type **1** and press Enter to update the panel.

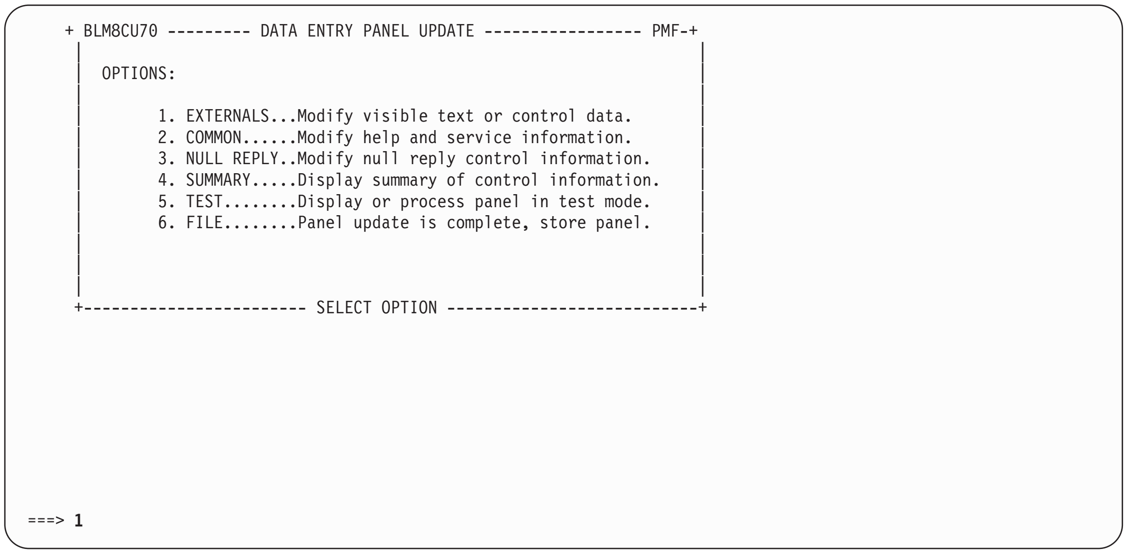


```
     + BLM8C000 -------- PANEL MODIFICATION FACILITY -------------- PMF-+
     |                                                                  |
     | OPTIONS:                                                         |
     |                                                                  |
     |       1. PANEL UPDATE.....Display, modify, or create panels.     |
     |       2. DICTIONARY.......Display or modify dictionary data.     |
     |       3. PANEL COPY.......Create new panel via panel copy.       |
     |       4. PANEL DELETE.....Delete existing panel.                 |
     |       5. REPORTS..........Panels, prefix, and other listings.    |
     |       6. PANEL SET........Create panels to be used for inquiry,  |
     |                           creation, update, etc. of records.     |
     |       7. PANEL LIST.......Display panel data set information.    |
     |                                                                  |
     +----------------------- SELECT OPTION ---------------------------+







===> 1
```

Type **1,blg0s010** on the control line and press Enter twice.


```
BLM8CU00                    PANEL NAME ENTRY                          UPDATE

Identify panel to be updated; cursor placement or input line entry allowed.




              1. Panel name.................<R> ________
              2. Data set definition label..... ________





  To enter the panel update dialog, press Enter without field modifications.








===> 1,blg0s010
```

Type **1** and press Enter to modify the externals, the visible text of BLG0S010.

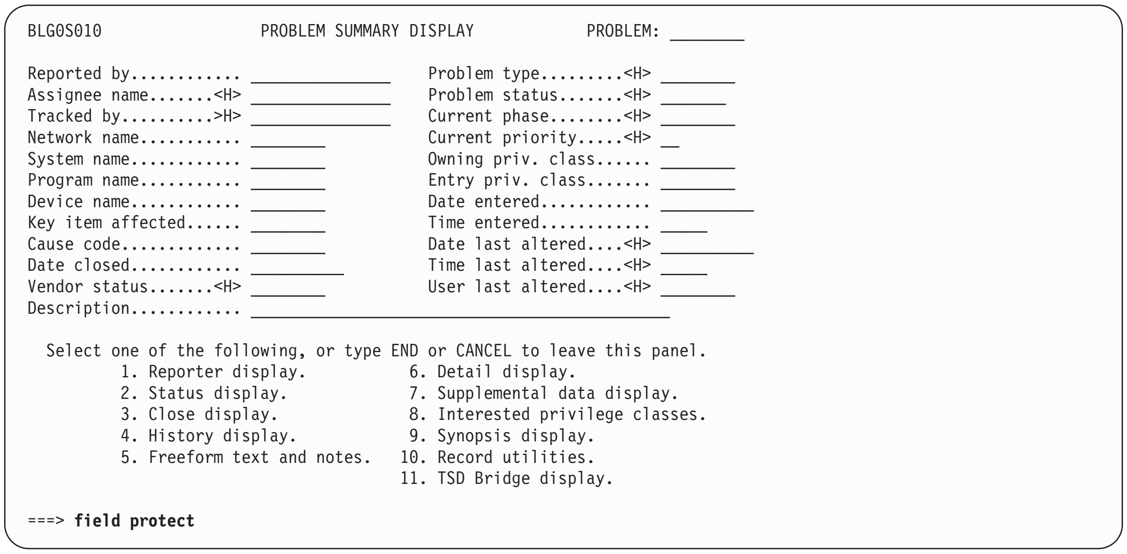

```
      + BLM8CU70 --------- DATA ENTRY PANEL UPDATE ----------------- PMF-+
      |                                                                  |
      |  OPTIONS:                                                        |
      |                                                                  |
      |        1. EXTERNALS...Modify visible text or control data.       |
      |        2. COMMON......Modify help and service information.       |
      |        3. NULL REPLY..Modify null reply control information.     |
      |        4. SUMMARY.....Display summary of control information.    |
      |        5. TEST........Display or process panel in test mode.     |
      |        6. FILE........Panel update is complete, store panel.     |
      |                                                                  |
      |                                                                  |
      |                                                                  |
      +------------------------ SELECT OPTION ---------------------------+








===> 1
```

On BLG0S010, the Problem Summary Display, add a selection for `TSD Bridge display`. In this example, do the following:

1. Type **field protect** on the command line, but do not yet press Enter.
2. Type **11. TSD Bridge display.** (or a number and text of your choosing) in the right column under **10. Record utilities.**
3. Position the cursor immediately to the left of the **11. TSD Bridge display.** field.

```
↓
 11. TSD Bridge display.
```

4. Press Enter.



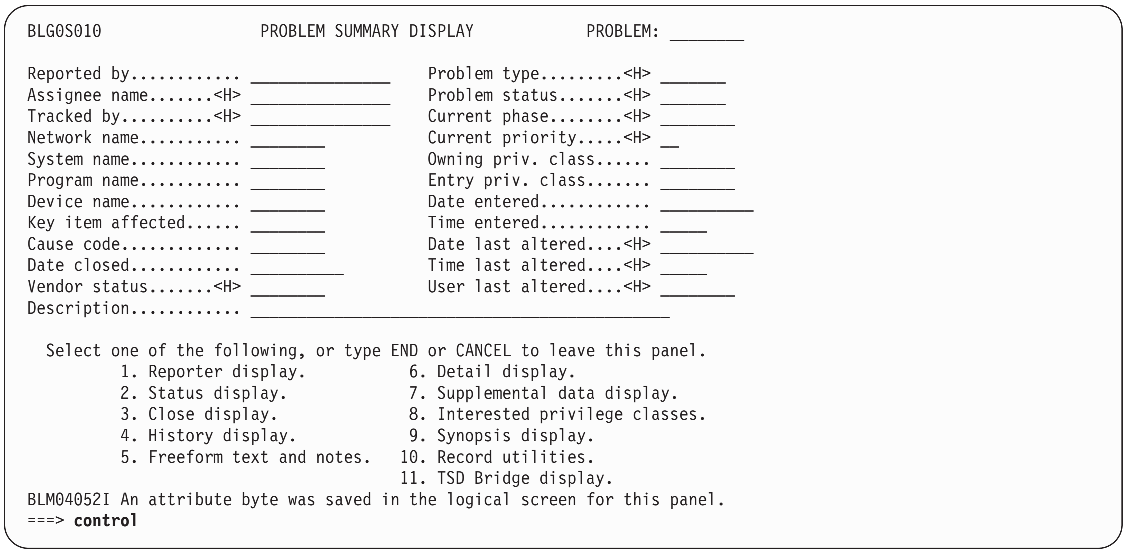
```
BLG0S010                PROBLEM SUMMARY DISPLAY            PROBLEM: ________

Reported by........... _______________     Problem type.........<H> ________
Assignee name.......<H> _______________    Problem status.......<H> _______
Tracked by..........>H> _______________    Current phase........<H> ________
Network name.......... ________            Current priority.....<H> __
System name........... ________            Owning priv. class...... ________
Program name.......... ________            Entry priv. class....... ________
Device name........... ________            Date entered........... __________
Key item affected..... ________            Time entered........... _____
Cause code............ ________            Date last altered....<H> __________
Date closed........... __________          Time last altered....<H> _____
Vendor status.......<H> ________           User last altered....<H> ________
Description........... _____________________________________________

  Select one of the following, or type END or CANCEL to leave this panel.
          1. Reporter display.          6. Detail display.
          2. Status display.            7. Supplemental data display.
          3. Close display.             8. Interested privilege classes.
          4. History display.           9. Synopsis display.
          5. Freeform text and notes.  10. Record utilities.
                                       11. TSD Bridge display.

===> field protect
```

After you press Enter, you will receive a message indicating that an attribute byte was saved for this field. Next, add control information for the **11. TSD Bridge display.** field.

1. Type **control** on the command line, but do not yet press Enter.
2. Again position the cursor immediately to the left of the **11. TSD Bridge display.** field.

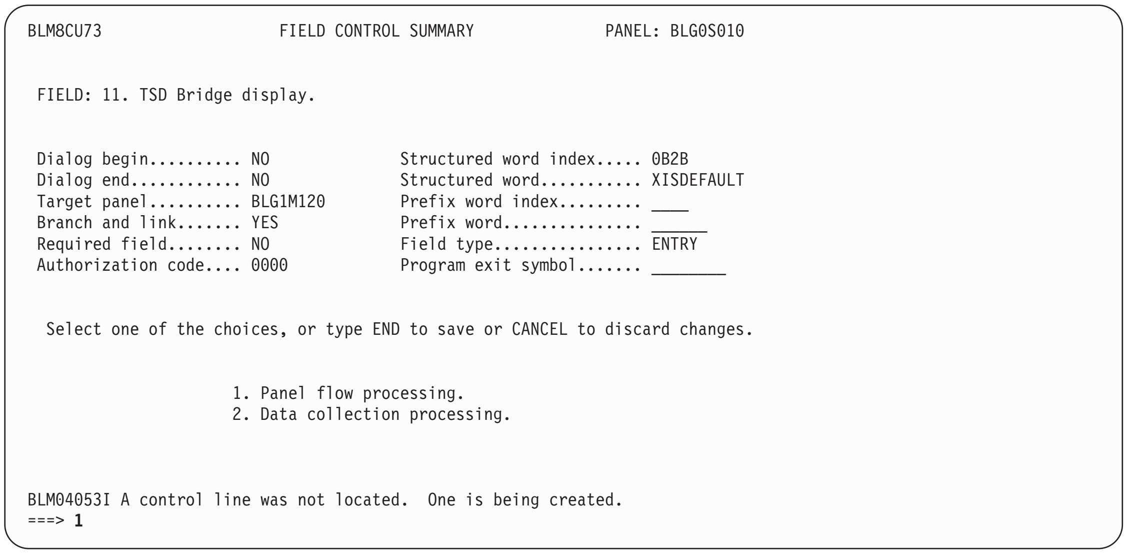
```
↓
 11. TSD Bridge display.
```

3. Press Enter.



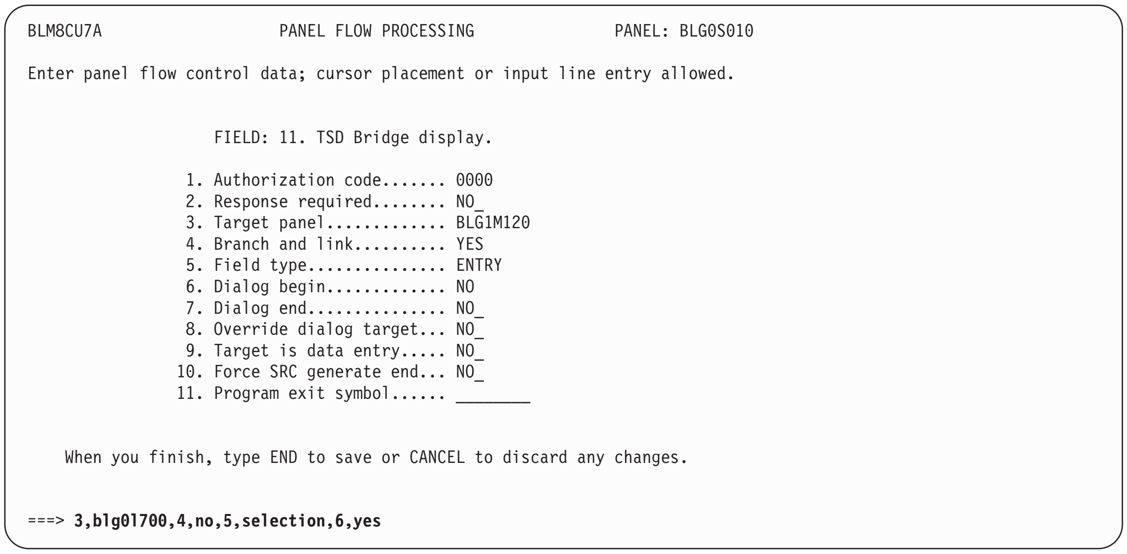
```
BLG0S010                PROBLEM SUMMARY DISPLAY            PROBLEM: ________

Reported by........... _______________     Problem type.........<H> _______
Assignee name.......<H> _______________    Problem status.......<H> _______
Tracked by..........<H> _______________    Current phase........<H> ________
Network name.......... ________            Current priority.....<H> __
System name........... ________            Owning priv. class...... ________
Program name.......... ________            Entry priv. class....... ________
Device name........... ________            Date entered........... __________
Key item affected..... ________            Time entered........... _____
Cause code............ ________            Date last altered....<H> __________
Date closed........... __________          Time last altered....<H> _____
Vendor status.......<H> ________           User last altered....<H> ________
Description........... _____________________________________________

  Select one of the following, or type END or CANCEL to leave this panel.
          1. Reporter display.          6. Detail display.
          2. Status display.            7. Supplemental data display.
          3. Close display.             8. Interested privilege classes.
          4. History display.           9. Synopsis display.
          5. Freeform text and notes.  10. Record utilities.
                                       11. TSD Bridge display.
BLM04052I An attribute byte was saved in the logical screen for this panel.
===> control
```

The Field Control Summary panel is displayed. Type **1** and press Enter to update Panel flow processing.



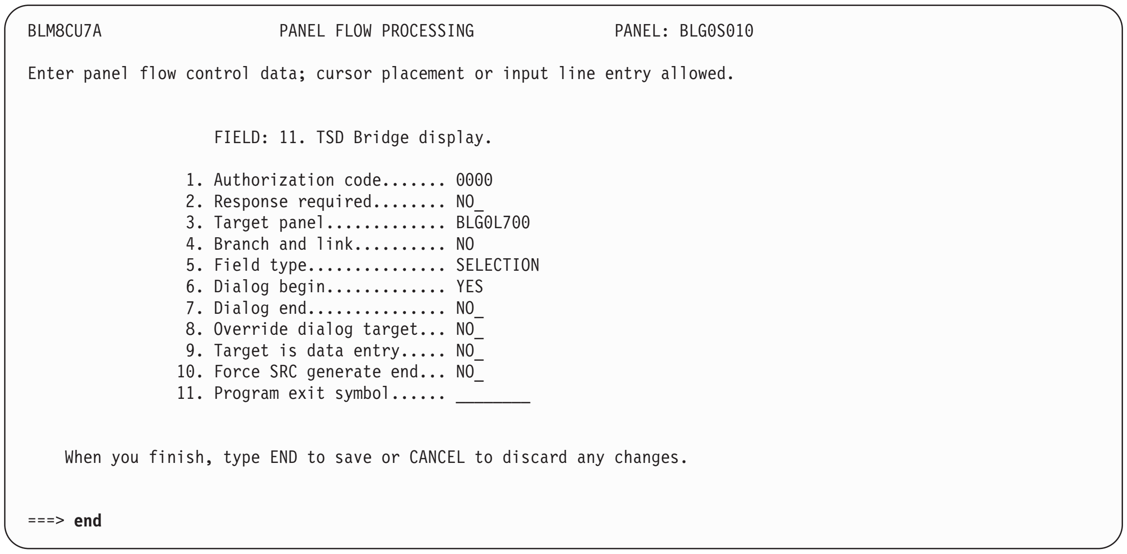
```
BLM8CU73                  FIELD CONTROL SUMMARY             PANEL: BLG0S010


 FIELD: 11. TSD Bridge display.


 Dialog begin......... NO               Structured word index..... 0B2B
 Dialog end........... NO               Structured word........... XISDEFAULT
 Target panel......... BLG1M120         Prefix word index......... ____
 Branch and link...... YES              Prefix word.............. ______
 Required field....... NO               Field type................ ENTRY
 Authorization code.... 0000            Program exit symbol....... ________


  Select one of the choices, or type END to save or CANCEL to discard changes.


                    1. Panel flow processing.
                    2. Data collection processing.



BLM04053I A control line was not located.  One is being created.
===> 1
```

Type **3,BLG0L700,4,no,5,selection,6,yes** and press Enter to set the **Target panel** to **BLG0L700**, **Branch and link** to **NO**, **Field type** to **Selection**, and **Dialog begin** to **YES**.



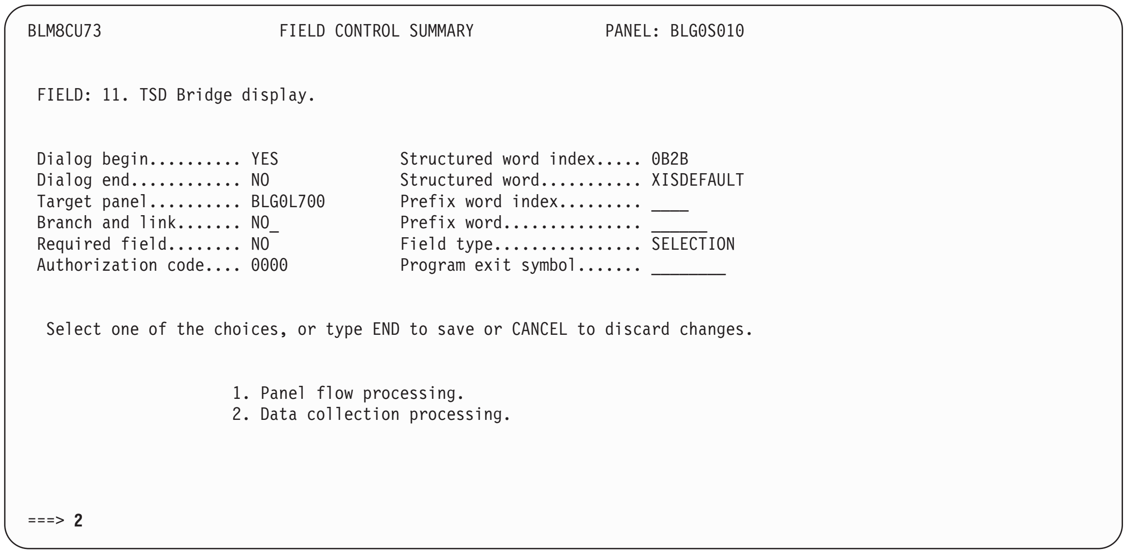
```
BLM8CU7A                  PANEL FLOW PROCESSING              PANEL: BLG0S010

Enter panel flow control data; cursor placement or input line entry allowed.


                   FIELD: 11. TSD Bridge display.

                1. Authorization code....... 0000
                2. Response required........ NO_
                3. Target panel............. BLG1M120
                4. Branch and link.......... YES
                5. Field type............... ENTRY
                6. Dialog begin............. NO
                7. Dialog end............... NO_
                8. Override dialog target... NO_
                9. Target is data entry..... NO_
               10. Force SRC generate end... NO_
               11. Program exit symbol...... ________


    When you finish, type END to save or CANCEL to discard any changes.


===> 3,blg0l700,4,no,5,selection,6,yes
```

Type **end** and press Enter to save your changes.



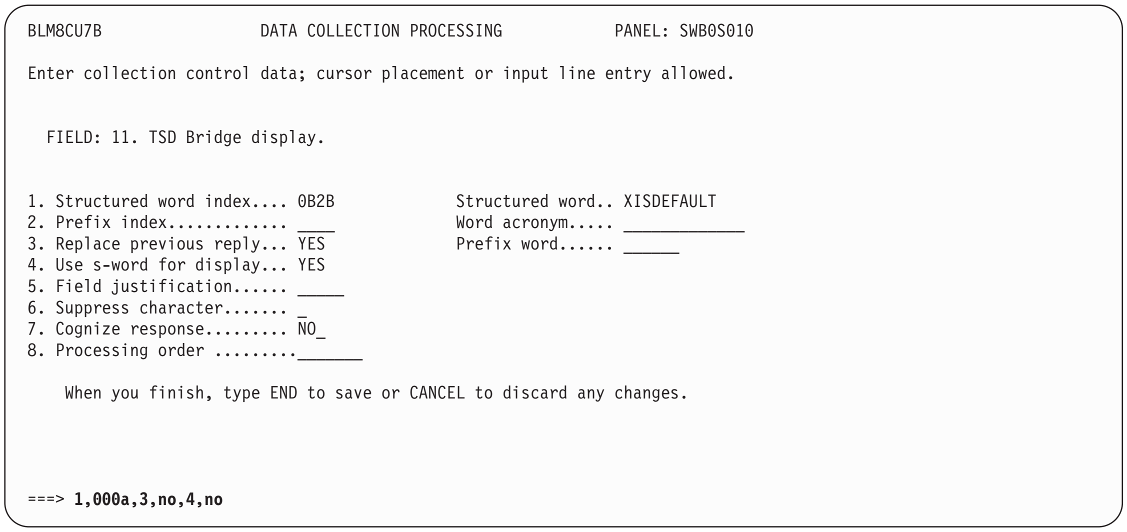
```
BLM8CU7A                  PANEL FLOW PROCESSING              PANEL: BLG0S010

Enter panel flow control data; cursor placement or input line entry allowed.


                    FIELD: 11. TSD Bridge display.

                 1. Authorization code....... 0000
                 2. Response required........ NO_
                 3. Target panel............. BLG0L700
                 4. Branch and link.......... NO
                 5. Field type............... SELECTION
                 6. Dialog begin............. YES
                 7. Dialog end............... NO_
                 8. Override dialog target... NO_
                 9. Target is data entry..... NO_
                10. Force SRC generate end... NO_
                11. Program exit symbol...... ________


    When you finish, type END to save or CANCEL to discard any changes.


===> end
```

From the Field Control Summary, type **2** and press Enter to update Data collection processing.



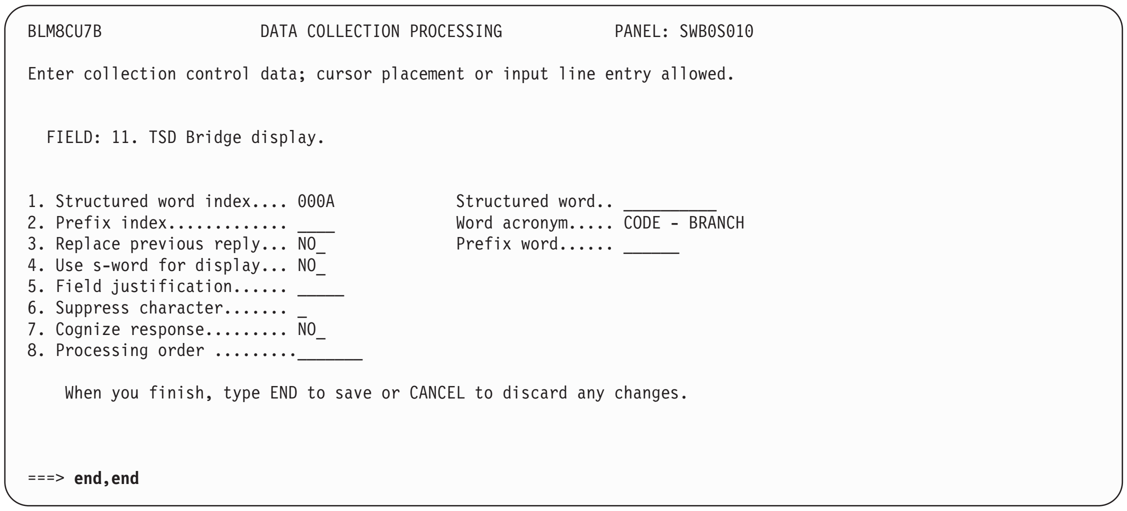
```
BLM8CU73                  FIELD CONTROL SUMMARY             PANEL: BLG0S010


 FIELD: 11. TSD Bridge display.


 Dialog begin.......... YES            Structured word index..... 0B2B
 Dialog end............ NO             Structured word........... XISDEFAULT
 Target panel.......... BLG0L700       Prefix word index......... ____
 Branch and link....... NO_            Prefix word.............. ______
 Required field........ NO             Field type................ SELECTION
 Authorization code.... 0000           Program exit symbol....... ________


  Select one of the choices, or type END to save or CANCEL to discard changes.


                      1. Panel flow processing.
                      2. Data collection processing.




===> 2
```

Type **1,000a,3,no,4,no** and press Enter to set the **Structured word index** to **000A**, set the **Replace previous reply** to **NO**, and set the **Use s-word for display** to **NO**.



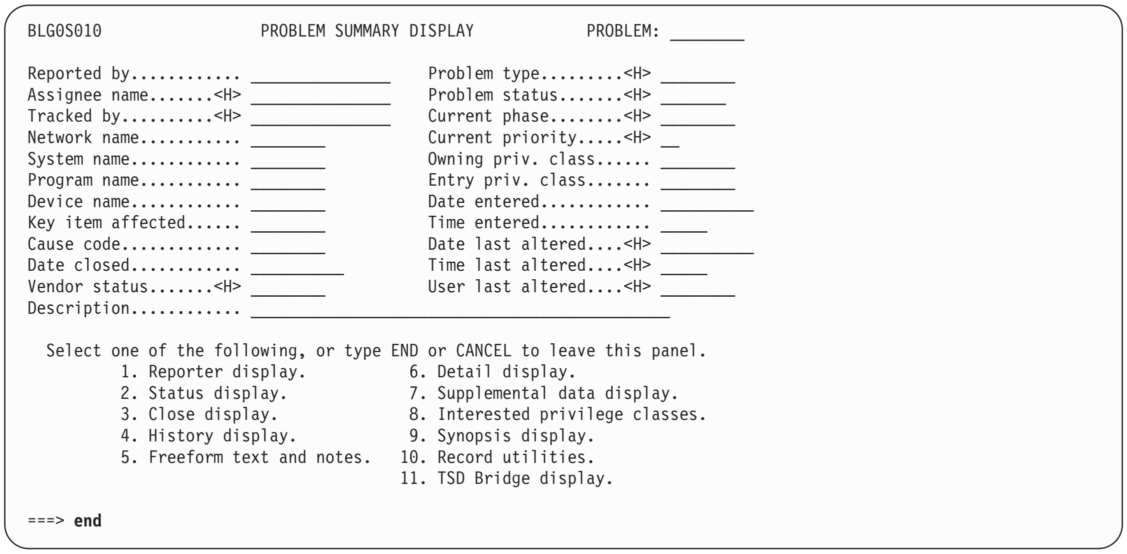
```
BLM8CU7B                DATA COLLECTION PROCESSING           PANEL: SWB0S010

Enter collection control data; cursor placement or input line entry allowed.


  FIELD: 11. TSD Bridge display.


1. Structured word index.... 0B2B             Structured word.. XISDEFAULT
2. Prefix index............ ____             Word acronym..... _____________
3. Replace previous reply... YES              Prefix word...... ______
4. Use s-word for display... YES
5. Field justification...... _____
6. Suppress character....... _
7. Cognize response......... NO_
8. Processing order ........._______

    When you finish, type END to save or CANCEL to discard any changes.



===> 1,000a,3,no,4,no
```

Type **end,end** and press Enter to save the changes and return to the externals for BLG0S010.



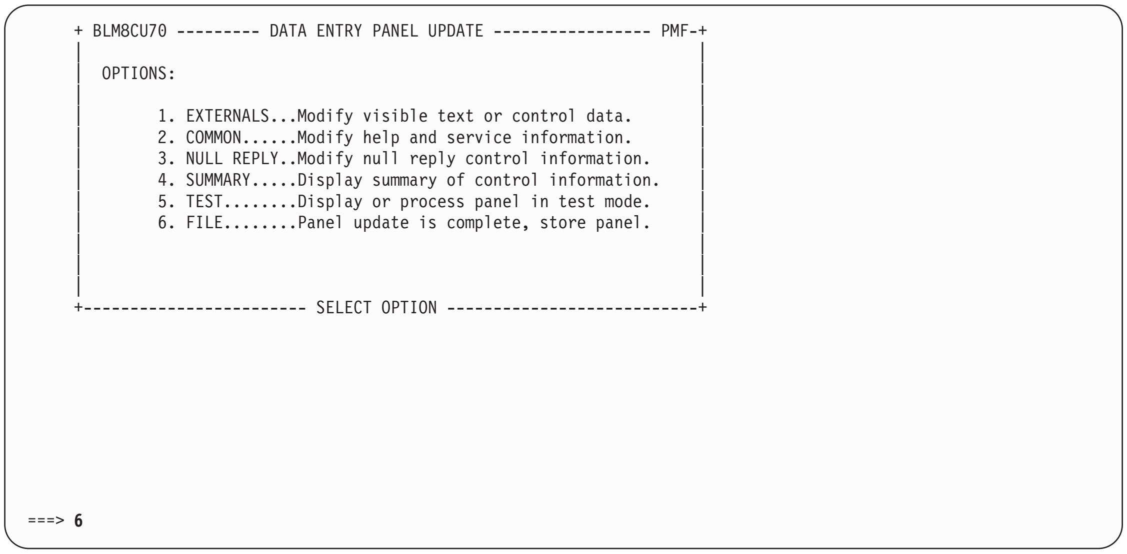
```
BLM8CU7B                DATA COLLECTION PROCESSING           PANEL: SWB0S010

Enter collection control data; cursor placement or input line entry allowed.


  FIELD: 11. TSD Bridge display.


1. Structured word index.... 000A             Structured word.. __________
2. Prefix index............ ____             Word acronym..... CODE - BRANCH
3. Replace previous reply... NO_              Prefix word...... ______
4. Use s-word for display... NO_
5. Field justification...... _____
6. Suppress character....... _
7. Cognize response......... NO_
8. Processing order ........._______

    When you finish, type END to save or CANCEL to discard any changes.



===> end,end
```

If panel BLG0S010 now appears satisfactory, type **end** and press Enter to return to the Data Entry Panel Update panel.



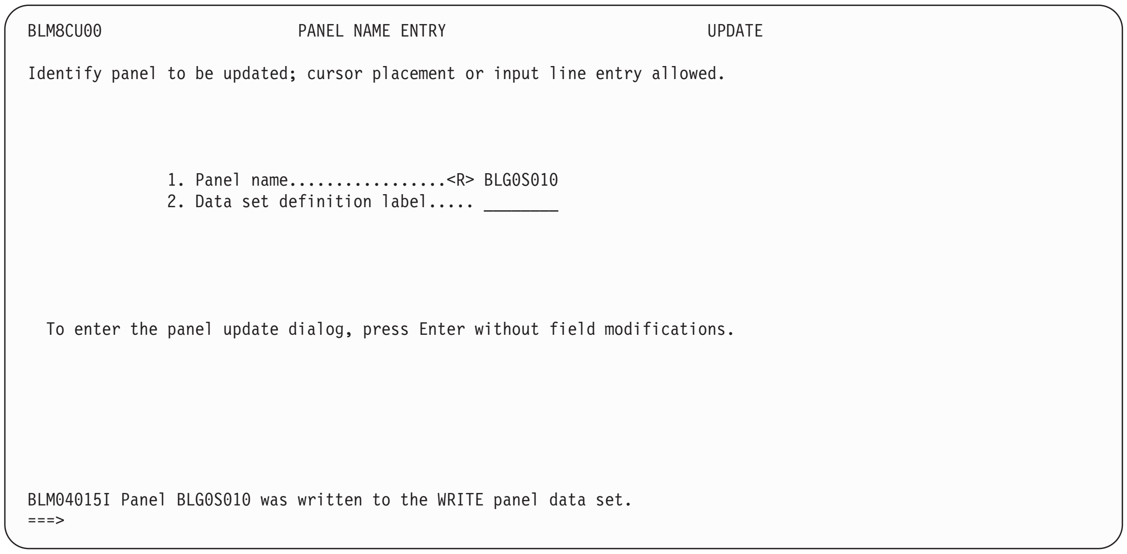
```
BLG0S010                PROBLEM SUMMARY DISPLAY           PROBLEM: ________

Reported by............ _______________     Problem type.........<H> ________
Assignee name.......<H> _______________     Problem status.......<H> _______
Tracked by..........<H> _______________     Current phase........<H> ________
Network name........... ________            Current priority.....<H> __
System name............ ________            Owning priv. class...... ________
Program name........... ________            Entry priv. class....... ________
Device name............ ________            Date entered............ __________
Key item affected...... ________            Time entered............ _____
Cause code............. ________            Date last altered....<H> __________
Date closed............ __________          Time last altered....<H> _____
Vendor status.......<H> ________            User last altered....<H> ________
Description............ _____________________________________________

  Select one of the following, or type END or CANCEL to leave this panel.
          1. Reporter display.          6. Detail display.
          2. Status display.            7. Supplemental data display.
          3. Close display.             8. Interested privilege classes.
          4. History display.           9. Synopsis display.
          5. Freeform text and notes.  10. Record utilities.
                                       11. TSD Bridge display.

===> end
```

Type **6** and press Enter to file the modified panel into your write panel data set.



```
       + BLM8CU70 --------- DATA ENTRY PANEL UPDATE ----------------- PMF-+
       |                                                                  |
       |  OPTIONS:                                                        |
       |                                                                  |
       |        1. EXTERNALS...Modify visible text or control data.       |
       |        2. COMMON......Modify help and service information.       |
       |        3. NULL REPLY..Modify null reply control information.     |
       |        4. SUMMARY.....Display summary of control information.    |
       |        5. TEST........Display or process panel in test mode.     |
       |        6. FILE........Panel update is complete, store panel.     |
       |                                                                  |
       |                                                                  |
       |                                                                  |
       +----------------------- SELECT OPTION ---------------------------+







===> 6
```

You will receive a message confirming that the panel was filed in the write panel data set. Copy the panel to a read panel data set when you are ready for those changes to go into effect.

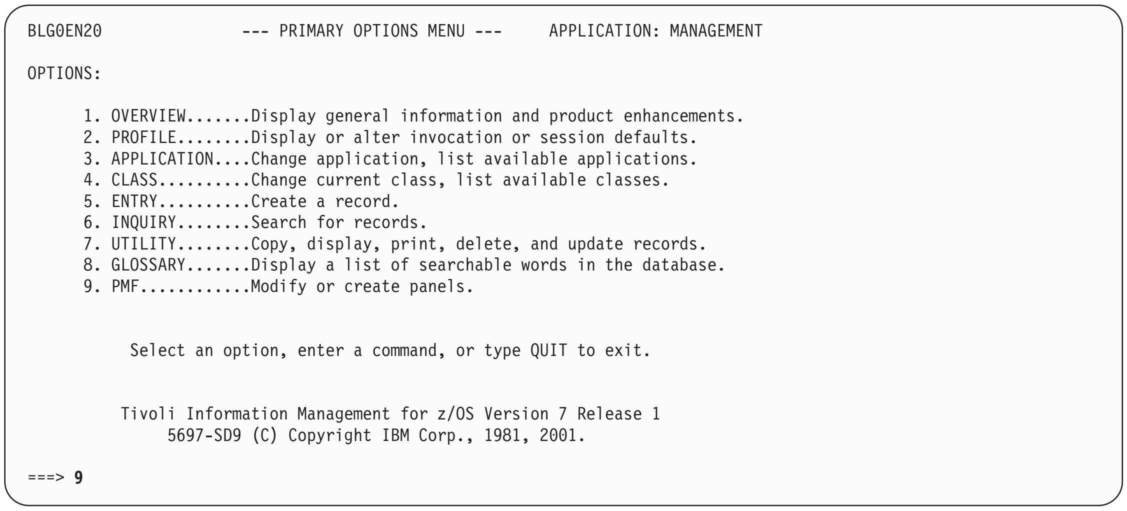


```
BLM8CU00                    PANEL NAME ENTRY                          UPDATE

Identify panel to be updated; cursor placement or input line entry allowed.




              1. Panel name.................<R> BLG0S010
              2. Data set definition label..... ________





  To enter the panel update dialog, press Enter without field modifications.







BLM04015I Panel BLG0S010 was written to the WRITE panel data set.
===>
```

## Updating Panel BLG0E090

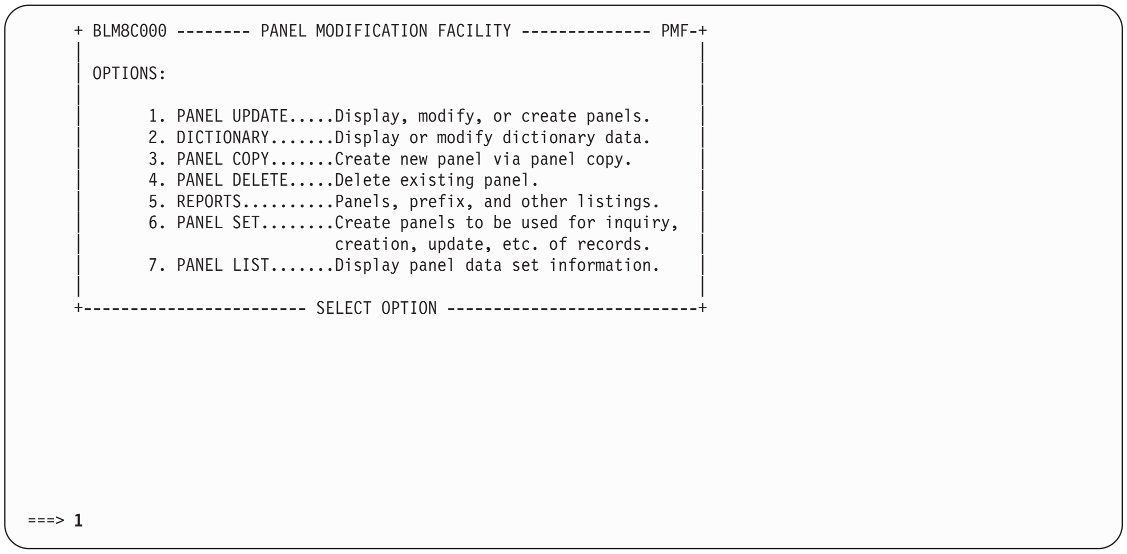
The Tivoli-supplied BLG0E090 contains **7. TSD Bridge data.** as a selection. If you use the Tivoli-supplied BLG0E090, you will not need to perform this modification. However, if you have customized BLG0E090 or you use a different Problem Inquiry Summary panel, you will need to update your modified panel to include a selection for TSD Bridge data. This selection will go to the Tivoli Service Desk Bridge Data Inquiry panel BLG0E790. On BLG0E790 you can enter search criteria for Tivoli Service Desk Bridge data. Following are the steps to add that selection.

From BLG0EN20, the Primary Options Menu, type **9** and press Enter to begin the PMF process.



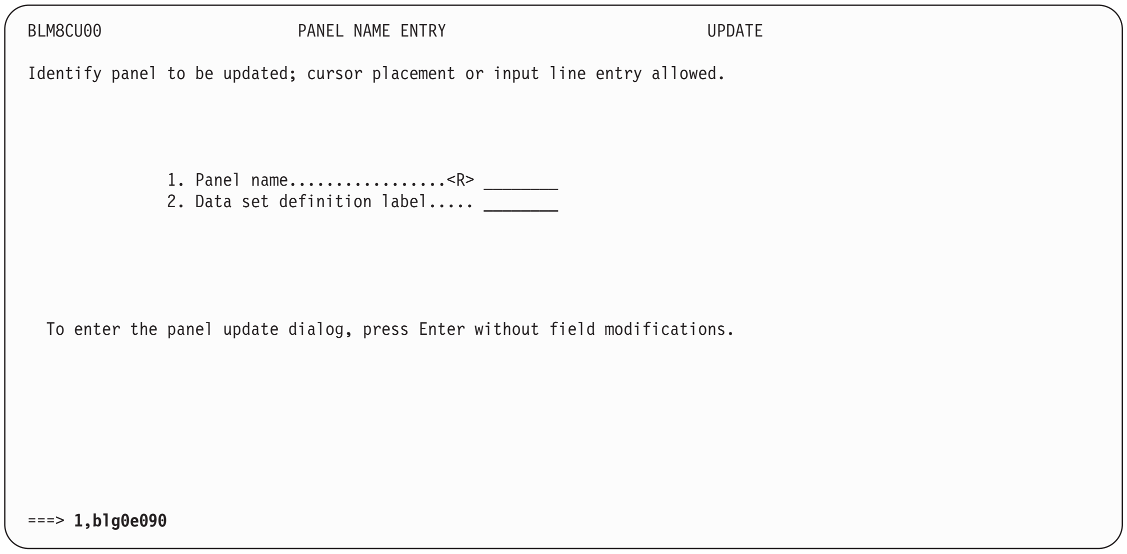
```
BLG0EN20              --- PRIMARY OPTIONS MENU ---     APPLICATION: MANAGEMENT

OPTIONS:

      1. OVERVIEW.......Display general information and product enhancements.
      2. PROFILE........Display or alter invocation or session defaults.
      3. APPLICATION....Change application, list available applications.
      4. CLASS..........Change current class, list available classes.
      5. ENTRY..........Create a record.
      6. INQUIRY........Search for records.
      7. UTILITY........Copy, display, print, delete, and update records.
      8. GLOSSARY.......Display a list of searchable words in the database.
      9. PMF............Modify or create panels.


           Select an option, enter a command, or type QUIT to exit.


          Tivoli Information Management for z/OS Version 7 Release 1
               5697-SD9 (C) Copyright IBM Corp., 1981, 2001.

===> 9
```

Type **1** and press Enter to update the panel.



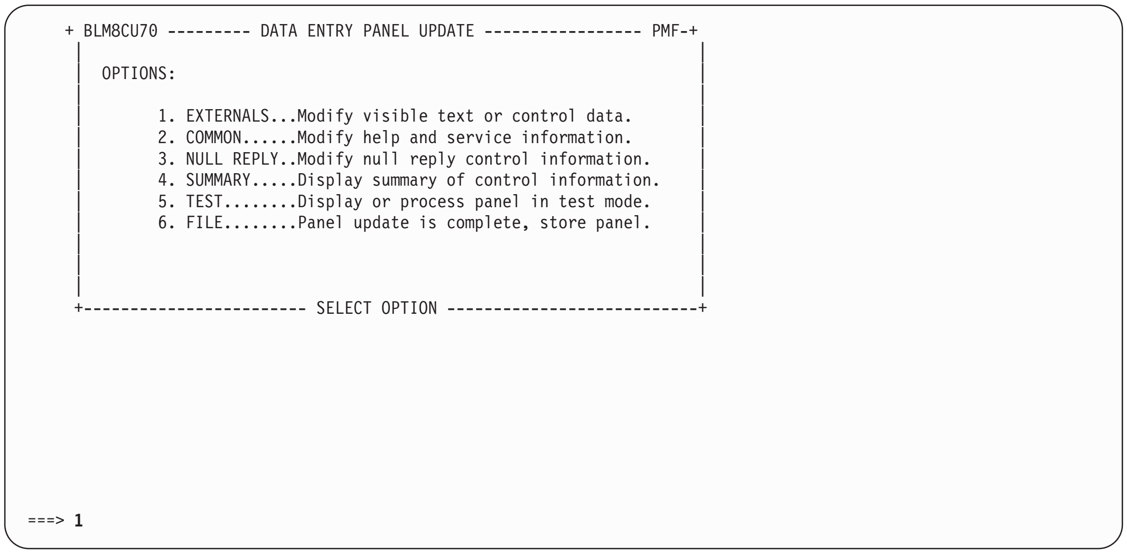
```
     + BLM8C000 -------- PANEL MODIFICATION FACILITY -------------- PMF-+
     |                                                                   |
     | OPTIONS:                                                          |
     |                                                                   |
     |       1. PANEL UPDATE.....Display, modify, or create panels.      |
     |       2. DICTIONARY.......Display or modify dictionary data.      |
     |       3. PANEL COPY.......Create new panel via panel copy.        |
     |       4. PANEL DELETE.....Delete existing panel.                  |
     |       5. REPORTS..........Panels, prefix, and other listings.     |
     |       6. PANEL SET........Create panels to be used for inquiry,   |
     |                           creation, update, etc. of records.      |
     |       7. PANEL LIST.......Display panel data set information.     |
     |                                                                   |
     +------------------------ SELECT OPTION ---------------------------+







===> 1
```

Type **1,blg0e090** on the control line and press Enter twice.



```
BLM8CU00                    PANEL NAME ENTRY                            UPDATE

Identify panel to be updated; cursor placement or input line entry allowed.



               1. Panel name................<R> ________
               2. Data set definition label..... ________




  To enter the panel update dialog, press Enter without field modifications.








===> 1,blg0e090
```

Type **1** and press Enter to modify the externals, the visible text of BLG0E090.



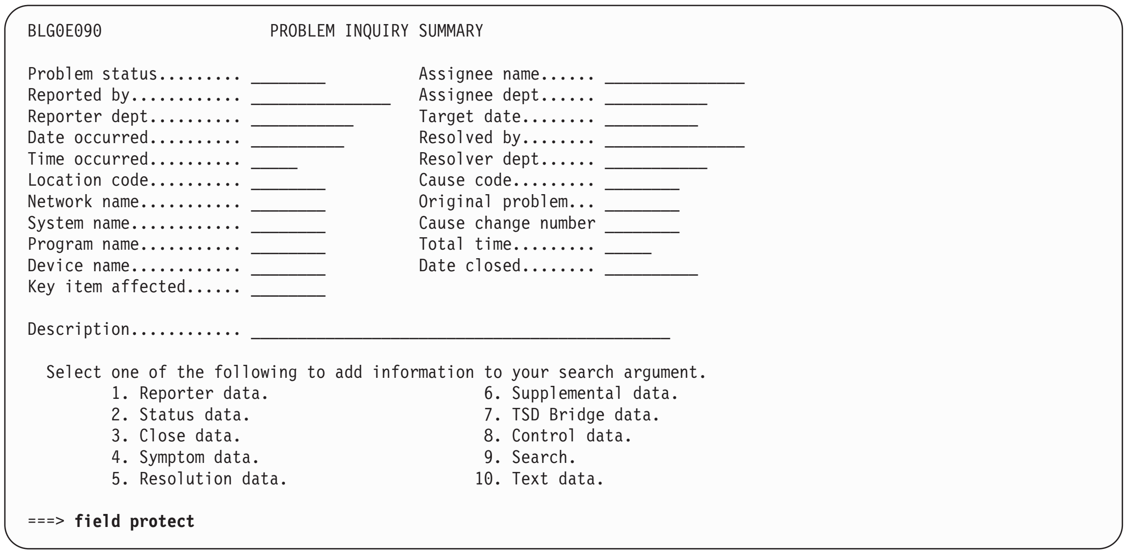
```
    + BLM8CU70 --------- DATA ENTRY PANEL UPDATE ----------------- PMF-+
     |                                                                  |
     |  OPTIONS:                                                        |
     |                                                                  |
     |        1. EXTERNALS...Modify visible text or control data.       |
     |        2. COMMON......Modify help and service information.       |
     |        3. NULL REPLY..Modify null reply control information.     |
     |        4. SUMMARY.....Display summary of control information.    |
     |        5. TEST........Display or process panel in test mode.     |
     |        6. FILE........Panel update is complete, store panel.     |
     |                                                                  |
     |                                                                  |
     |                                                                  |
     +------------------------ SELECT OPTION ---------------------------+





===> 1
```

On BLG0E090, the Problem Inquiry Summary, add a selection for **TSD Bridge data.** (or similar words). In this example, do the following:

1. Type **field protect** on the command line, but do not yet press Enter.
2. Type **7. TSD Bridge data.** (or a number and text of your choosing) in the right column under **6. Supplemental data.**
3. Position the cursor immediately to the left of the **7. TSD Bridge data.** field.

```
 ↓
  7. TSD Bridge data.
```

4. Press Enter.



```
BLG0E090                 PROBLEM INQUIRY SUMMARY

Problem status......... ________         Assignee name...... ______________
Reported by............ _______________  Assignee dept...... __________
Reporter dept.......... ___________      Target date........ _________
Date occurred.......... __________       Resolved by........ ______________
Time occurred.......... _____            Resolver dept...... __________
Location code.......... ________         Cause code......... ________
Network name........... ________         Original problem... ________
System name............ ________         Cause change number ________
Program name........... ________         Total time......... _____
Device name............ ________         Date closed........ _________
Key item affected...... ________

Description............ _____________________________________________

  Select one of the following to add information to your search argument.
         1. Reporter data.                    6. Supplemental data.
         2. Status data.                      7. TSD Bridge data.
         3. Close data.                       8. Control data.
         4. Symptom data.                     9. Search.
         5. Resolution data.                 10. Text data.

===> field protect
```

After you press Enter, you will receive a message indicating that an attribute byte was saved for this field. Next, add control information for the **7. TSD Bridge data.** field.

1. Type **control** on the command line, but do not yet press Enter.
2. Again position the cursor immediately to the left of the **7. TSD Bridge data.** field.

```
↓
 7. TSD Bridge display.
```

3. Press Enter.

```
BLG0E090                 PROBLEM INQUIRY SUMMARY

Problem status......... ________         Assignee name...... _______________
Reported by............ _______________  Assignee dept...... ___________
Reporter dept.......... ___________      Target date........ __________
Date occurred.......... __________       Resolved by........ _______________
Time occurred.......... _____            Resolver dept...... ___________
Location code.......... ________         Cause code......... ________
Network name........... ________         Original problem... ________
System name............ ________         Cause change number ________
Program name........... ________         Total time......... _____
Device name............ ________         Date closed........ __________
Key item affected...... ________

Description............ _____________________________________________

  Select one of the following to add information to your search argument.
         1. Reporter data.                    6. Supplemental data.
         2. Status data.                      7. TSD Bridge data.
         3. Close data.                       8. Control data.
         4. Symptom data.                     9. Search.
         5. Resolution data.                 10. Text data.
BLM04025I An attribute byte was saved in the logical screen for this panel.
===> control
```

The Field Control Summary panel is displayed. Type **1** and press Enter to update Panel flow processing.

```
BLM8CU73                  FIELD CONTROL SUMMARY             PANEL: BLG0E090


 FIELD: 7. TSD Bridge data.


 Dialog begin.......... NO              Structured word index..... 0B2B
 Dialog end............ NO              Structured word........... XISDEFAULT
 Target panel.......... BLG1M120        Prefix word index......... ____
 Branch and link....... YES             Prefix word.............. ______
 Required field........ NO              Field type................ ENTRY
 Authorization code.... 0000            Program exit symbol....... ________


  Select one of the choices, or type END to save or CANCEL to discard changes.


                     1. Panel flow processing.
                     2. Data collection processing.



BLM04053I A control line was not located.  One is being created.
===> 1
```

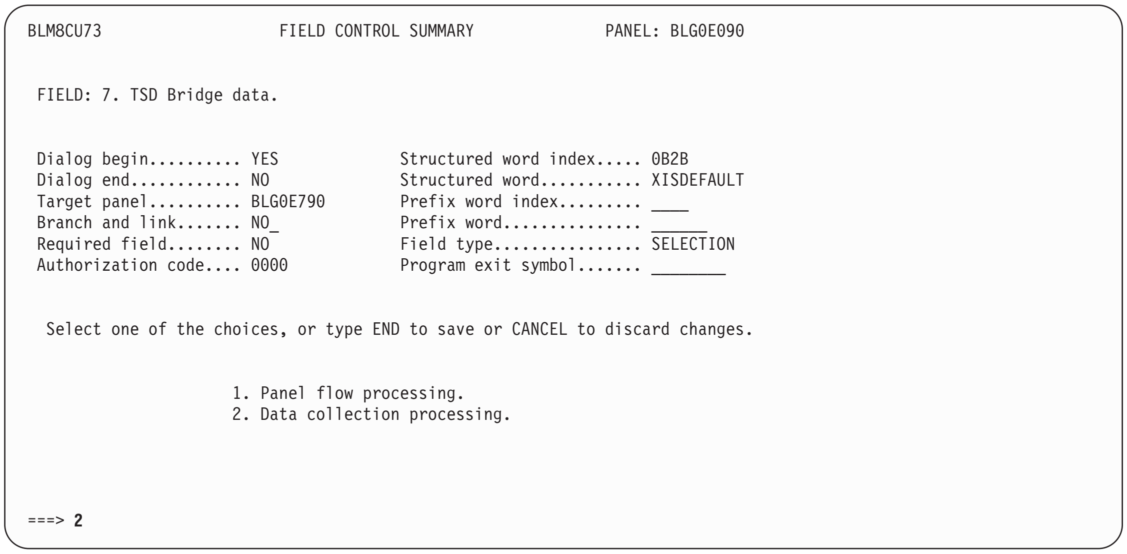
Type **3,BLG0E790,4,no,5,selection,6,yes** and press Enter to set the **Target panel** to **BLG0E790**, **Branch and link** to **NO**, **Field type** to **Selection**, and **Dialog begin** to **YES**.

```
BLM8CU7A                  PANEL FLOW PROCESSING             PANEL: BLG0E090

Enter panel flow control data; cursor placement or input line entry allowed.


                   FIELD: 7. TSD Bridge data.

                1. Authorization code....... 0000
                2. Response required........ NO_
                3. Target panel............. BLG1M120
                4. Branch and link.......... YES
                5. Field type............... ENTRY
                6. Dialog begin............. NO
                7. Dialog end............... NO_
                8. Override dialog target... NO_
                9. Target is data entry..... NO_
               10. Force SRC generate end... NO_
               11. Program exit symbol...... ________


    When you finish, type END to save or CANCEL to discard any changes.


===> 3,blg0E790,4,no,5,selection,6,yes
```

Type **end** and press Enter to save your changes.

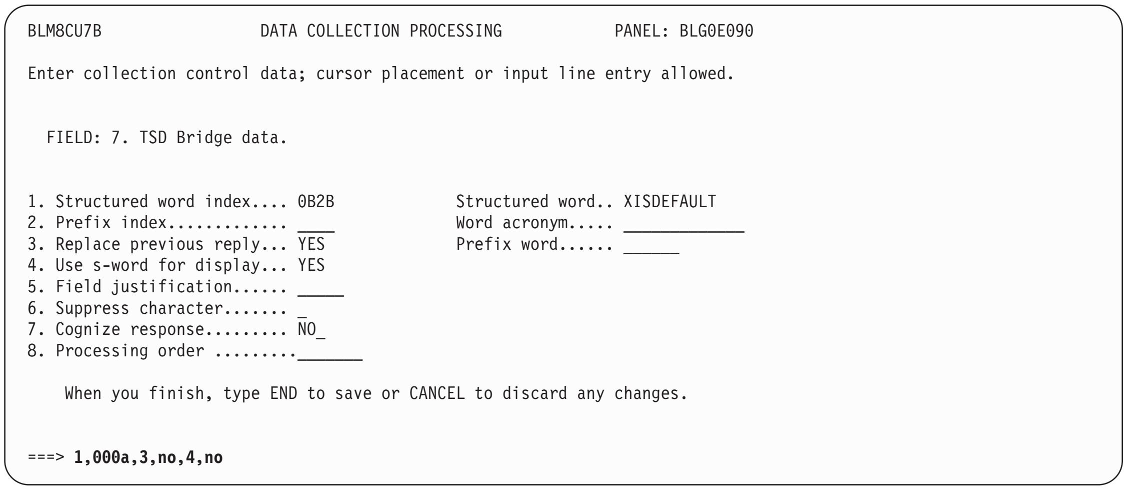
```
BLM8CU7A                  PANEL FLOW PROCESSING             PANEL: BLG0E090

Enter panel flow control data; cursor placement or input line entry allowed.


                   FIELD: 7. TSD Bridge data.

                1. Authorization code....... 0000
                2. Response required........ NO_
                3. Target panel............. BLG0E790
                4. Branch and link.......... NO
                5. Field type............... SELECTION
                6. Dialog begin............. YES
                7. Dialog end............... NO_
                8. Override dialog target... NO_
                9. Target is data entry..... NO_
               10. Force SRC generate end... NO_
               11. Program exit symbol...... ________


    When you finish, type END to save or CANCEL to discard any changes.


===> end
```

On the Field Control Summary, type **2** and press Enter to update Data collection processing.



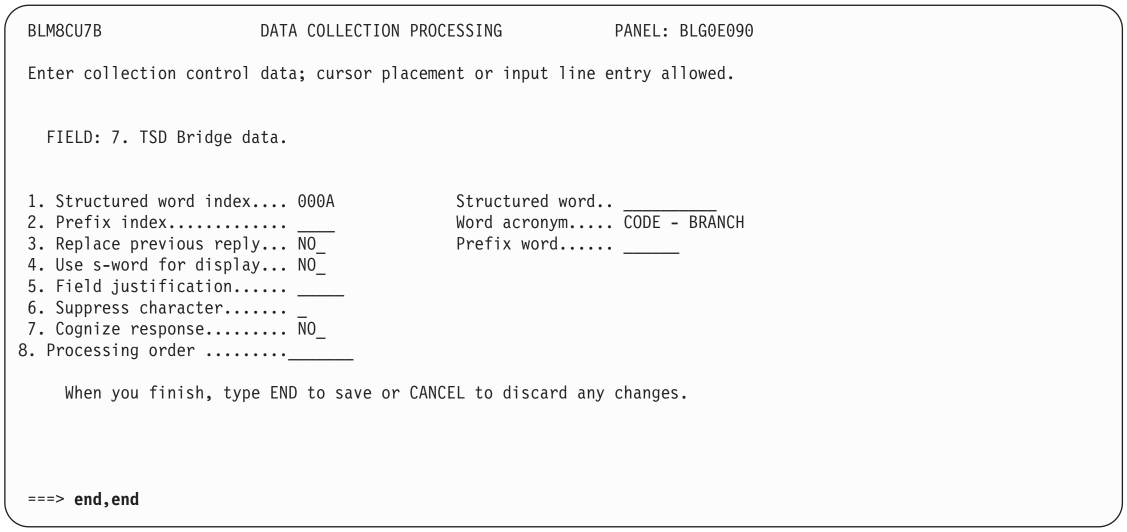
```
BLM8CU73                 FIELD CONTROL SUMMARY            PANEL: BLG0E090


 FIELD: 7. TSD Bridge data.


 Dialog begin.......... YES            Structured word index..... 0B2B
 Dialog end............ NO             Structured word........... XISDEFAULT
 Target panel.......... BLG0E790       Prefix word index......... ____
 Branch and link....... NO_            Prefix word.............. ______
 Required field........ NO             Field type................ SELECTION
 Authorization code.... 0000           Program exit symbol....... ________


  Select one of the choices, or type END to save or CANCEL to discard changes.


                    1. Panel flow processing.
                    2. Data collection processing.




===> 2
```

Type **1,000a,3,no,4,no** and press Enter to set the **Structured word index** to **000A**, set the **Replace previous reply** to **NO**, and set the **Use s-word for display** to **NO**.



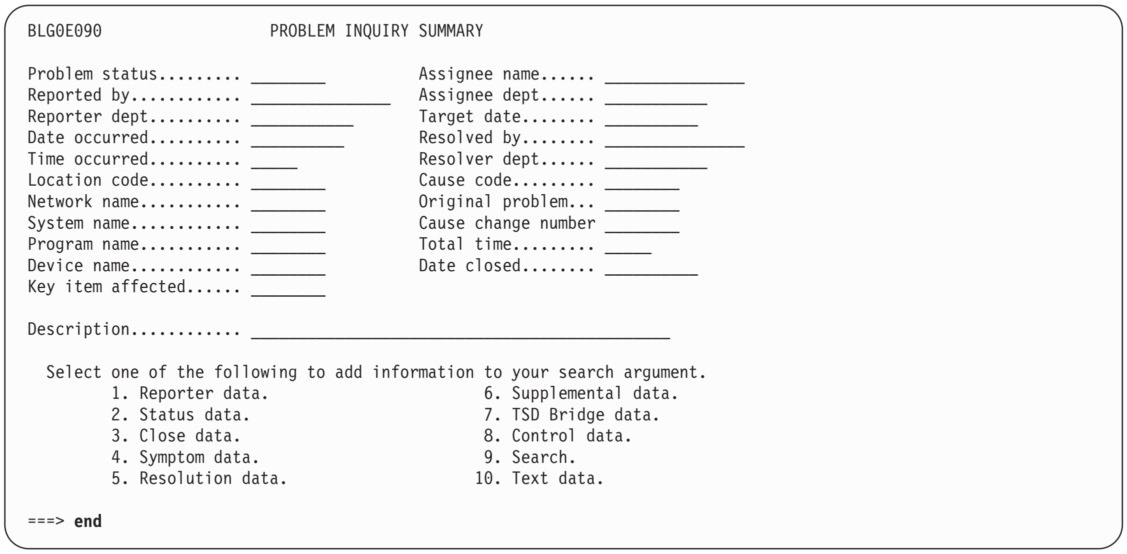
```
BLM8CU7B                DATA COLLECTION PROCESSING           PANEL: BLG0E090

Enter collection control data; cursor placement or input line entry allowed.


  FIELD: 7. TSD Bridge data.


1. Structured word index.... 0B2B            Structured word.. XISDEFAULT
2. Prefix index............ ____             Word acronym..... ____________
3. Replace previous reply... YES             Prefix word...... ______
4. Use s-word for display... YES
5. Field justification...... _____
6. Suppress character....... _
7. Cognize response......... NO_
8. Processing order ........._______

    When you finish, type END to save or CANCEL to discard any changes.


===> 1,000a,3,no,4,no
```

Type **end,end** and press Enter to save the changes and return to the externals for BLG0E090.



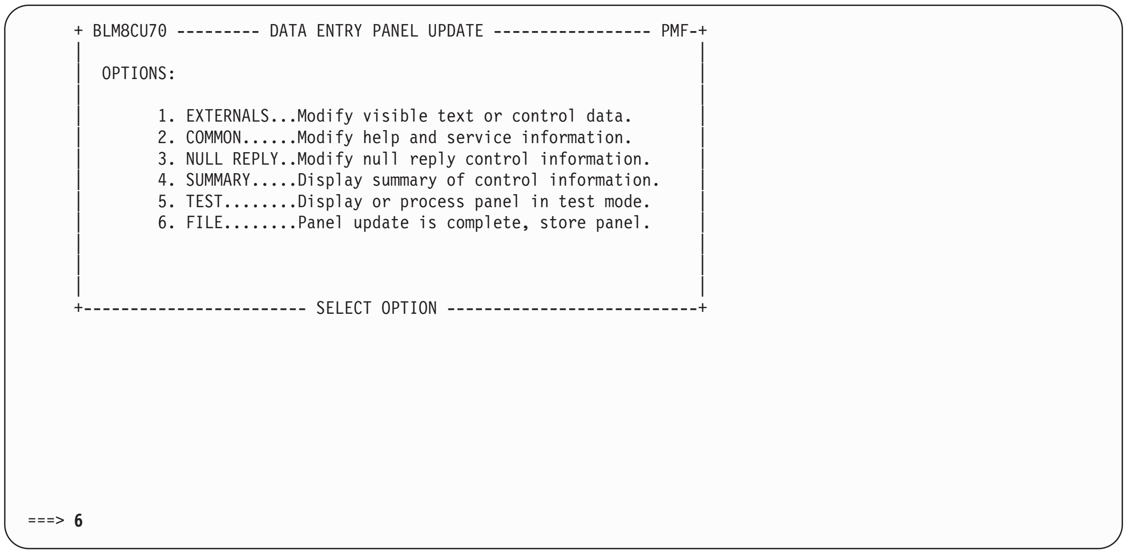
```
BLM8CU7B                DATA COLLECTION PROCESSING           PANEL: BLG0E090

Enter collection control data; cursor placement or input line entry allowed.


  FIELD: 7. TSD Bridge data.


1. Structured word index.... 000A            Structured word.. __________
2. Prefix index............ ____             Word acronym..... CODE - BRANCH
3. Replace previous reply... NO_             Prefix word...... ______
4. Use s-word for display... NO_
5. Field justification...... _____
6. Suppress character....... _
7. Cognize response......... NO_
8. Processing order .........______

    When you finish, type END to save or CANCEL to discard any changes.




===> end,end
```

If panel BLG0E090 now appears satisfactory, type **end** and press Enter.



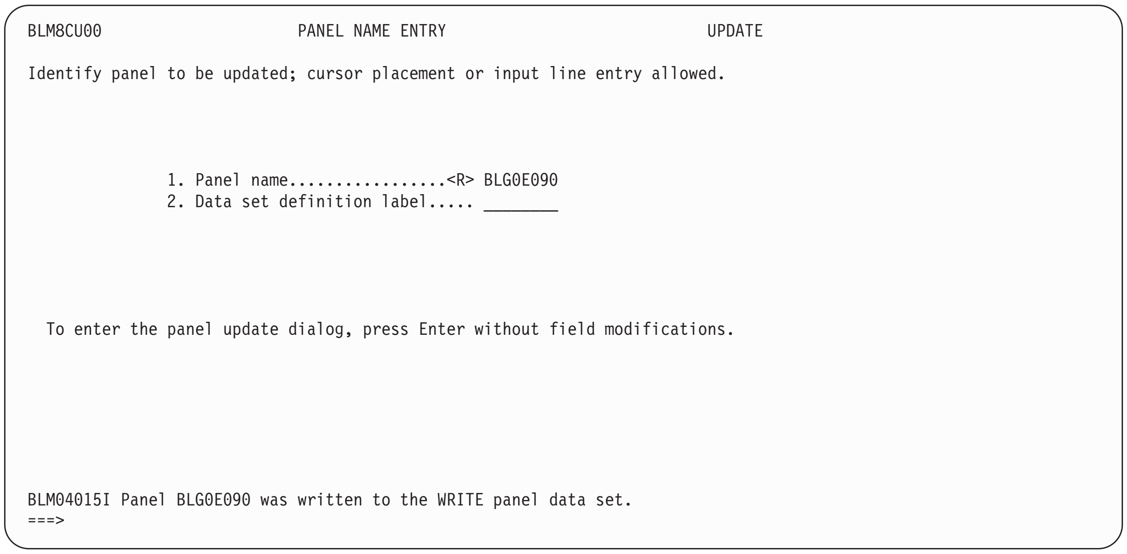
```
BLG0E090                 PROBLEM INQUIRY SUMMARY

Problem status......... ________          Assignee name...... _______________
Reported by............ _______________   Assignee dept...... ___________
Reporter dept.......... ___________       Target date........ __________
Date occurred.......... __________        Resolved by........ _______________
Time occurred.......... _____             Resolver dept...... ___________
Location code.......... ________          Cause code......... ________
Network name........... ________          Original problem... ________
System name............ ________          Cause change number ________
Program name........... ________          Total time......... _____
Device name............ ________          Date closed........ __________
Key item affected...... ________

Description............ _____________________________________________

  Select one of the following to add information to your search argument.
         1. Reporter data.                    6. Supplemental data.
         2. Status data.                      7. TSD Bridge data.
         3. Close data.                       8. Control data.
         4. Symptom data.                     9. Search.
         5. Resolution data.                 10. Text data.

===> end
```

Type **6** and press Enter to file the modified panel into your write panel data set.



```
    + BLM8CU70 --------- DATA ENTRY PANEL UPDATE ---------------- PMF-+
    |                                                                 |
    |  OPTIONS:                                                       |
    |                                                                 |
    |        1. EXTERNALS...Modify visible text or control data.      |
    |        2. COMMON......Modify help and service information.      |
    |        3. NULL REPLY..Modify null reply control information.    |
    |        4. SUMMARY.....Display summary of control information.   |
    |        5. TEST........Display or process panel in test mode.    |
    |        6. FILE........Panel update is complete, store panel.    |
    |                                                                 |
    |                                                                 |
    |                                                                 |
    +------------------------ SELECT OPTION --------------------------+









===> 6
```

You will receive a message confirming that the panel was filed in the write panel data set. Copy the panel to a read panel data set when you are ready for those changes to go into effect.

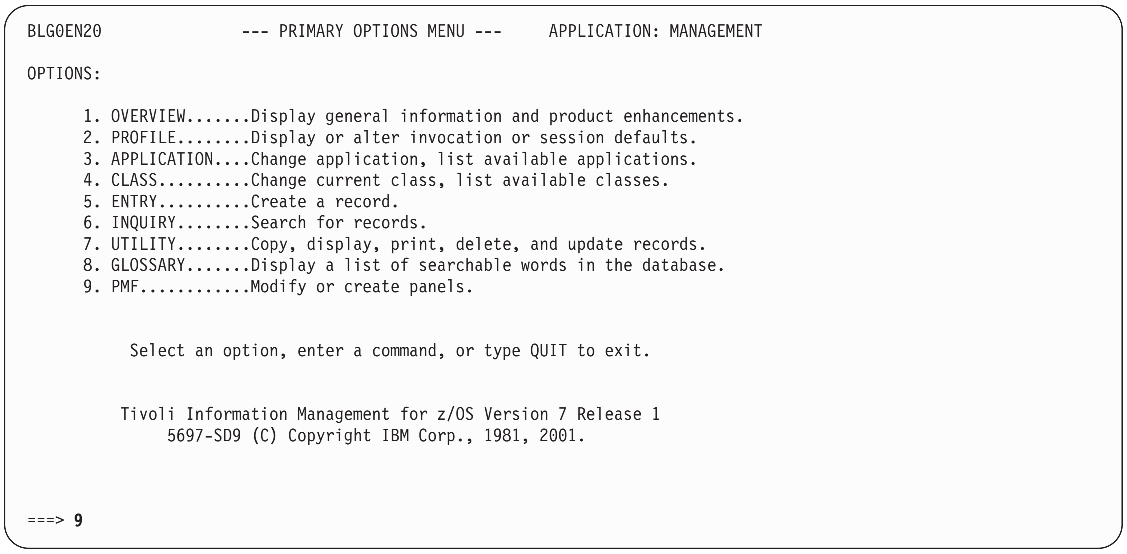


```
BLM8CU00                    PANEL NAME ENTRY                             UPDATE

Identify panel to be updated; cursor placement or input line entry allowed.




               1. Panel name.................<R> BLG0E090
               2. Data set definition label..... ________





  To enter the panel update dialog, press Enter without field modifications.







BLM04015I Panel BLG0E090 was written to the WRITE panel data set.
===>
```

## Updating Panel BLG1A111

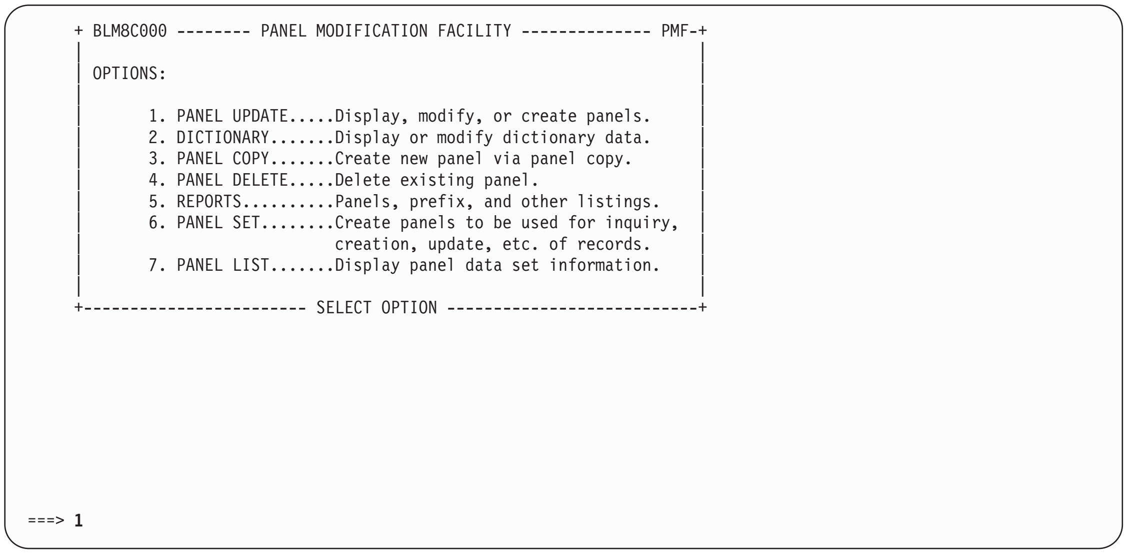
BLG1A111 is the panel that is invoked when a problem record is filed. If you have not previously modified panel BLG1A111, use the Panel Modification Facility (PMF) to copy BLG1A11Z from the base panel data set into the read panel data set and rename it to BLG1A111. A process for doing this is described in “Copying Panel BLG1A11Z” on page 345. If you have previously modified panel BLG1A111 for other purposes, you should use PMF to change it to invoke the TSX BLGTSDPT. Following is a process for doing that.

From BLG0EN20, the Primary Options Menu, type **9** and press Enter to begin the PMF process.



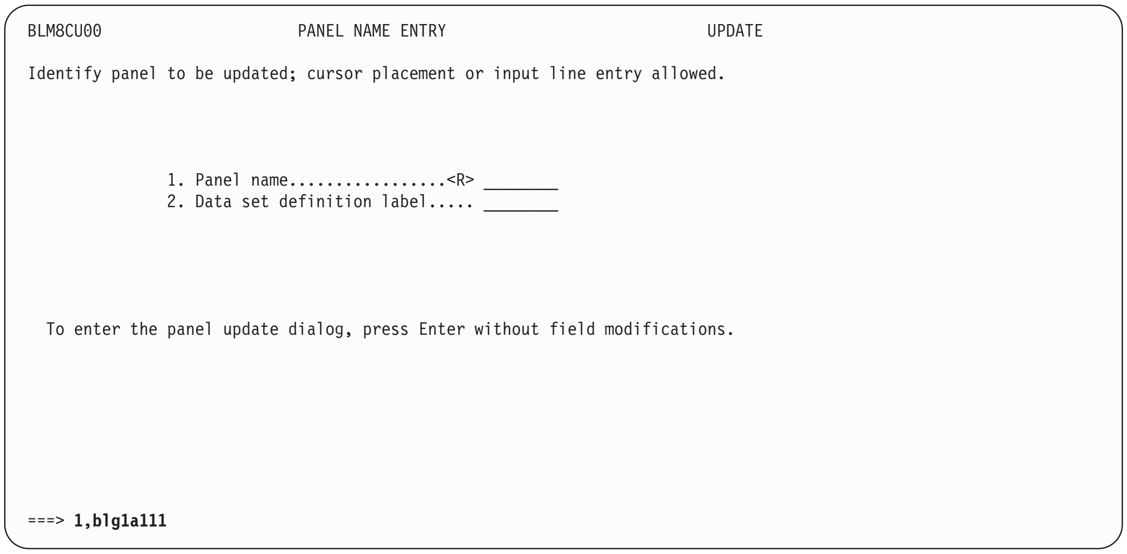
```
BLG0EN20              --- PRIMARY OPTIONS MENU ---     APPLICATION: MANAGEMENT

OPTIONS:

      1. OVERVIEW.......Display general information and product enhancements.
      2. PROFILE........Display or alter invocation or session defaults.
      3. APPLICATION....Change application, list available applications.
      4. CLASS..........Change current class, list available classes.
      5. ENTRY..........Create a record.
      6. INQUIRY........Search for records.
      7. UTILITY........Copy, display, print, delete, and update records.
      8. GLOSSARY.......Display a list of searchable words in the database.
      9. PMF............Modify or create panels.


           Select an option, enter a command, or type QUIT to exit.


          Tivoli Information Management for z/OS Version 7 Release 1
               5697-SD9 (C) Copyright IBM Corp., 1981, 2001.



===> 9
```

Type **1** and press Enter to update the panel.



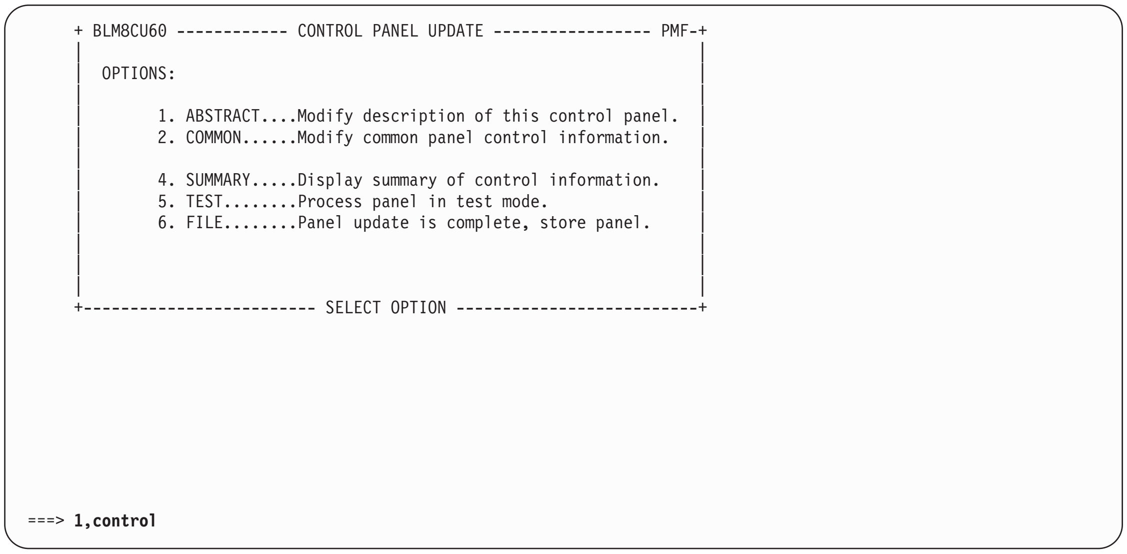
```
       + BLM8C000 -------- PANEL MODIFICATION FACILITY -------------- PMF-+
       |                                                                  |
       | OPTIONS:                                                         |
       |                                                                  |
       |       1. PANEL UPDATE.....Display, modify, or create panels.     |
       |       2. DICTIONARY.......Display or modify dictionary data.     |
       |       3. PANEL COPY.......Create new panel via panel copy.       |
       |       4. PANEL DELETE.....Delete existing panel.                 |
       |       5. REPORTS..........Panels, prefix, and other listings.    |
       |       6. PANEL SET........Create panels to be used for inquiry,  |
       |                           creation, update, etc. of records.     |
       |       7. PANEL LIST.......Display panel data set information.    |
       |                                                                  |
       +----------------------- SELECT OPTION ---------------------------+







===> 1
```

Type **1,blg1a111** and press Enter twice.



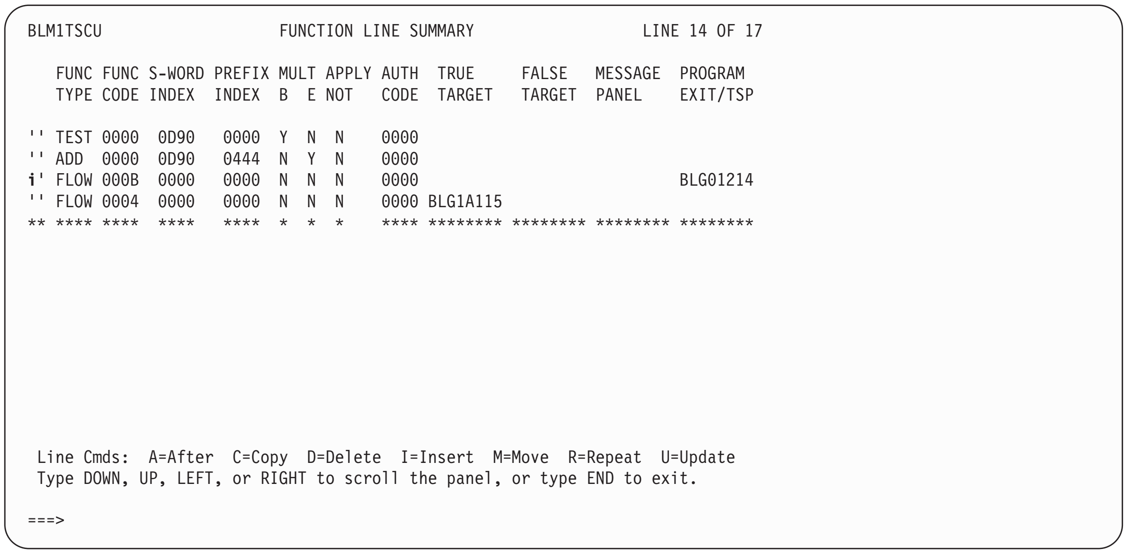
```
BLM8CU00                    PANEL NAME ENTRY                            UPDATE

Identify panel to be updated; cursor placement or input line entry allowed.




               1. Panel name.................<R> ________
               2. Data set definition label..... ________





  To enter the panel update dialog, press Enter without field modifications.








===> 1,blg1a111
```

Type **1,control** and press Enter to go to the Function Line Summary for this panel.



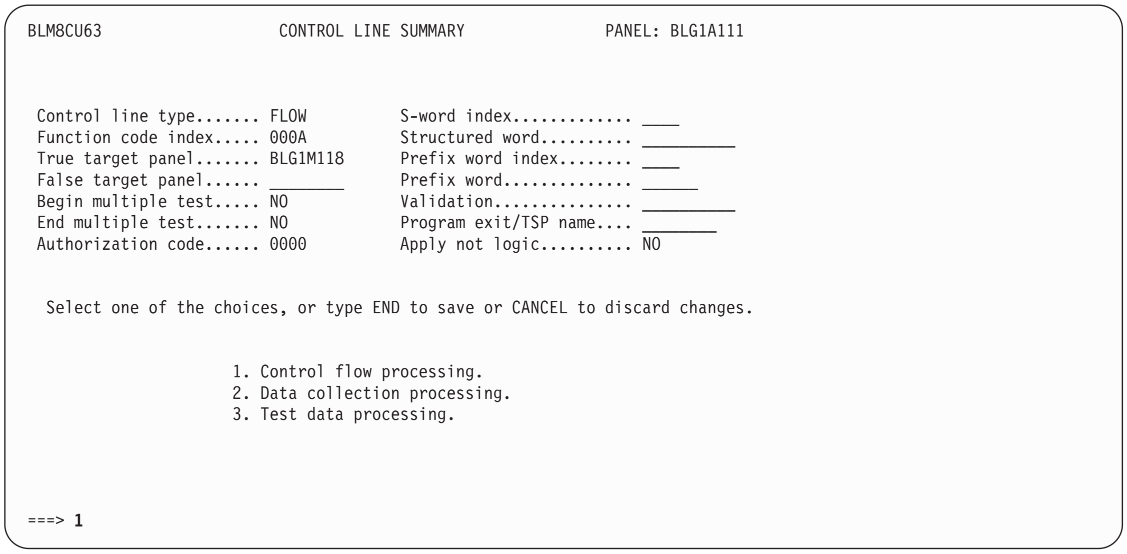
```
     + BLM8CU60 ------------ CONTROL PANEL UPDATE ----------------- PMF-+
     |                                                                  |
     |  OPTIONS:                                                        |
     |                                                                  |
     |        1. ABSTRACT....Modify description of this control panel.  |
     |        2. COMMON......Modify common panel control information.   |
     |                                                                  |
     |        4. SUMMARY.....Display summary of control information.    |
     |        5. TEST........Process panel in test mode.                |
     |        6. FILE........Panel update is complete, store panel.     |
     |                                                                  |
     |                                                                  |
     |                                                                  |
     +------------------------- SELECT OPTION --------------------------+







===> 1,control
```

Type **i** on the FLOW line that calls program exit BLG01214 to insert a line after it. Then press Enter. You may need to scroll down to find this line.



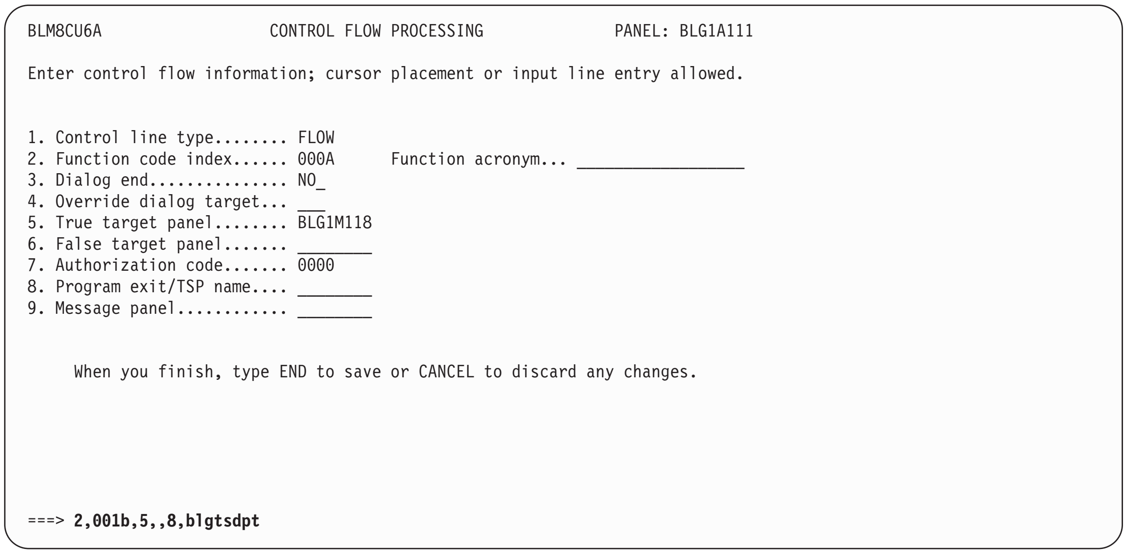
```
BLM1TSCU                  FUNCTION LINE SUMMARY                 LINE 14 OF 17

   FUNC FUNC S-WORD PREFIX MULT APPLY AUTH  TRUE     FALSE   MESSAGE  PROGRAM
   TYPE CODE INDEX  INDEX  B  E NOT   CODE  TARGET   TARGET  PANEL    EXIT/TSP

'' TEST 0000  0D90   0000  Y  N  N    0000
'' ADD  0000  0D90   0444  N  Y  N    0000
i' FLOW 000B  0000   0000  N  N  N    0000                            BLG01214
'' FLOW 0004  0000   0000  N  N  N    0000 BLG1A115
** **** ****  ****   ****  *  *  *    **** ******** ******** ******** ********

 Line Cmds:  A=After  C=Copy  D=Delete  I=Insert  M=Move  R=Repeat  U=Update
 Type DOWN, UP, LEFT, or RIGHT to scroll the panel, or type END to exit.

===>
```

Type **1** and press Enter to specify Control flow processing.



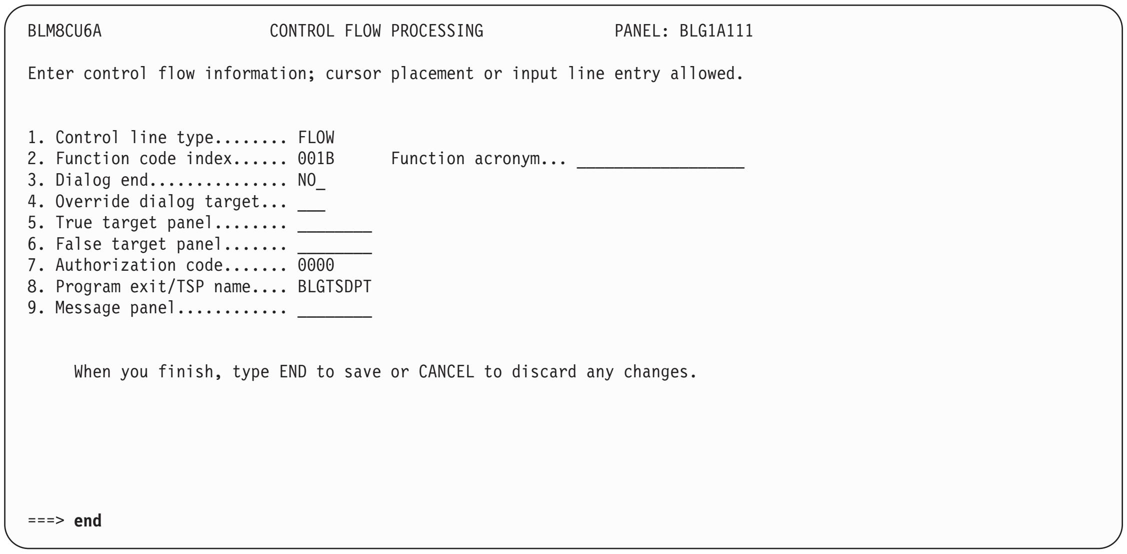
```
BLM8CU63                  CONTROL LINE SUMMARY              PANEL: BLG1A111



 Control line type....... FLOW          S-word index............. ____
 Function code index..... 000A          Structured word.......... __________
 True target panel....... BLG1M118      Prefix word index........ ____
 False target panel...... ________      Prefix word.............. ______
 Begin multiple test..... NO            Validation............... __________
 End multiple test....... NO            Program exit/TSP name.... ________
 Authorization code...... 0000          Apply not logic.......... NO


  Select one of the choices, or type END to save or CANCEL to discard changes.


                     1. Control flow processing.
                     2. Data collection processing.
                     3. Test data processing.



===> 1
```

Type **2,001b,5,,8,blgtsdpt** and press Enter to set the **Function code index** to 001B, clear the **True target panel** field, and set the **Program exit/TSP name** to BLGTSDPT.



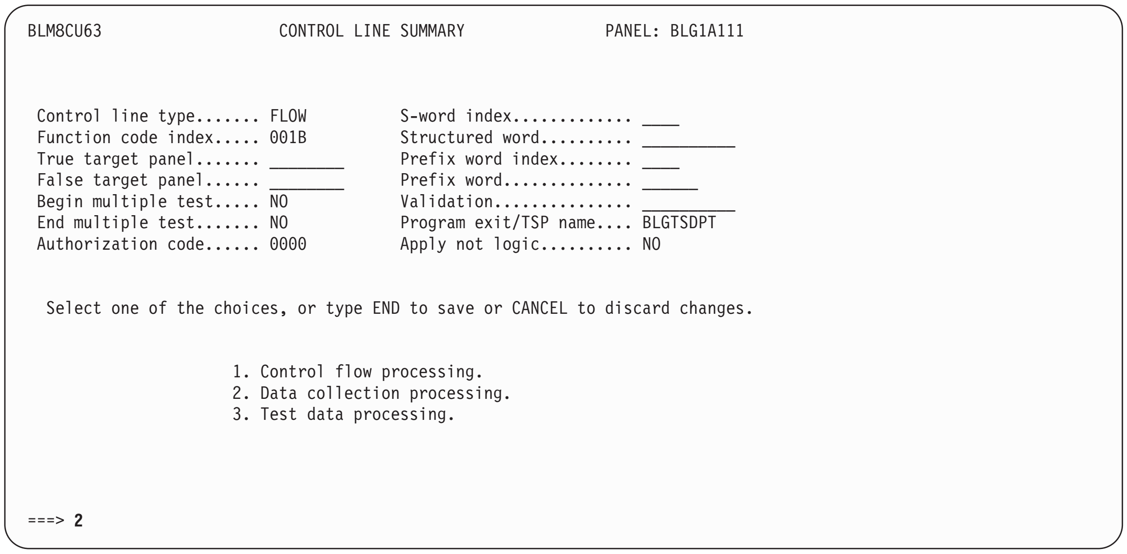
```
BLM8CU6A                CONTROL FLOW PROCESSING             PANEL: BLG1A111

Enter control flow information; cursor placement or input line entry allowed.


1. Control line type........ FLOW
2. Function code index...... 000A      Function acronym... __________________
3. Dialog end............... NO_
4. Override dialog target... ___
5. True target panel........ BLG1M118
6. False target panel....... ________
7. Authorization code....... 0000
8. Program exit/TSP name.... ________
9. Message panel............ ________


     When you finish, type END to save or CANCEL to discard any changes.






===> 2,001b,5,,8,blgtsdpt
```

Type **end** and press Enter to return to the Control Line Summary panel.



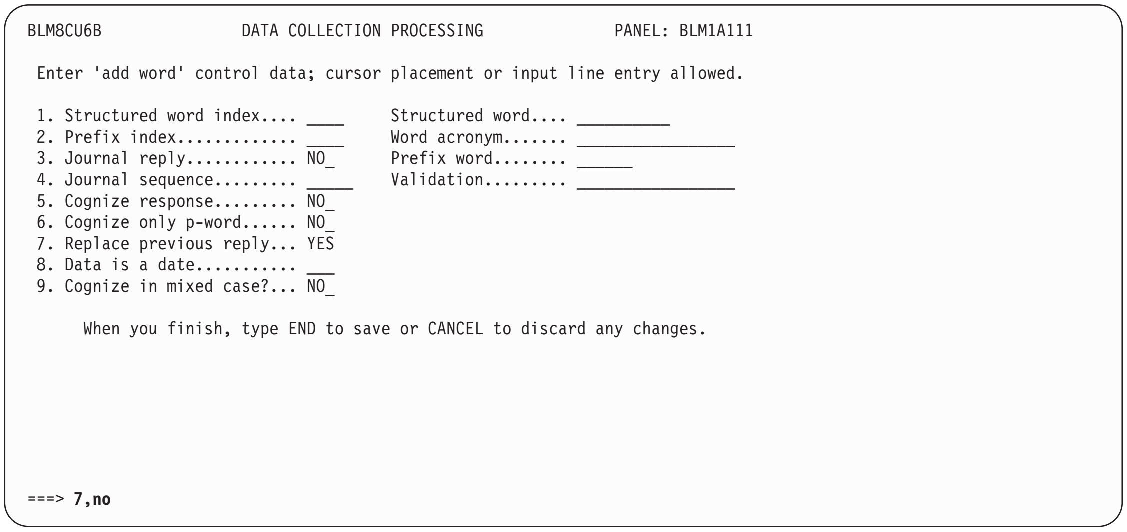
```
BLM8CU6A                CONTROL FLOW PROCESSING             PANEL: BLG1A111

Enter control flow information; cursor placement or input line entry allowed.


1. Control line type........ FLOW
2. Function code index...... 001B      Function acronym... __________________
3. Dialog end............... NO_
4. Override dialog target... ___
5. True target panel........ ________
6. False target panel....... ________
7. Authorization code....... 0000
8. Program exit/TSP name.... BLGTSDPT
9. Message panel............ ________


     When you finish, type END to save or CANCEL to discard any changes.






===> end
```

Type **2** and press Enter to modify the Data collection processing.



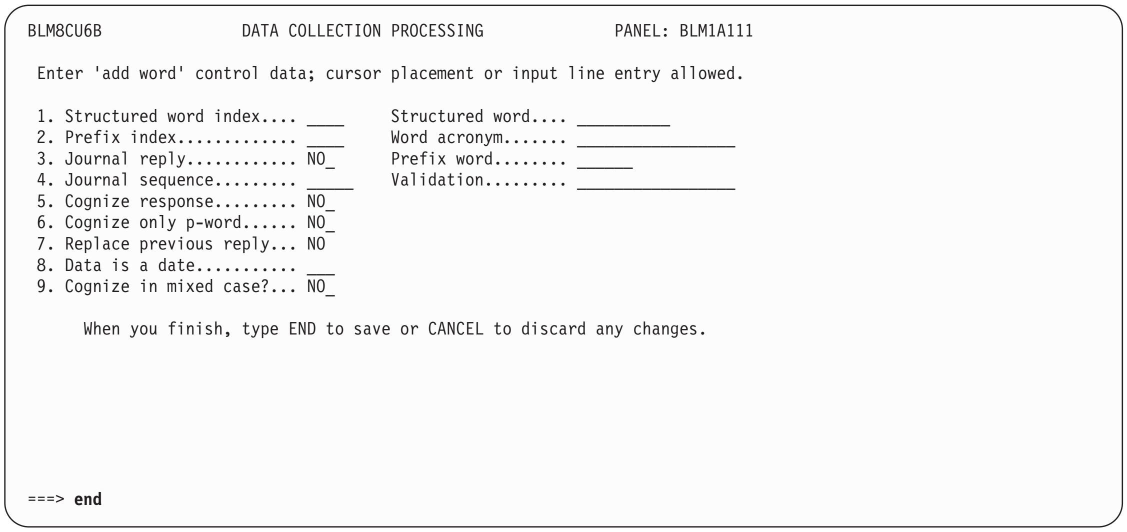
```
BLM8CU63                 CONTROL LINE SUMMARY             PANEL: BLG1A111



 Control line type....... FLOW         S-word index............ ____
 Function code index..... 001B         Structured word.......... __________
 True target panel....... ________     Prefix word index........ ____
 False target panel...... ________     Prefix word............. ______
 Begin multiple test..... NO           Validation.............. __________
 End multiple test....... NO           Program exit/TSP name.... BLGTSDPT
 Authorization code...... 0000         Apply not logic.......... NO


  Select one of the choices, or type END to save or CANCEL to discard changes.


                    1. Control flow processing.
                    2. Data collection processing.
                    3. Test data processing.





===> 2
```

Type **7,no** and press Enter to set **Replace previous reply** to NO.



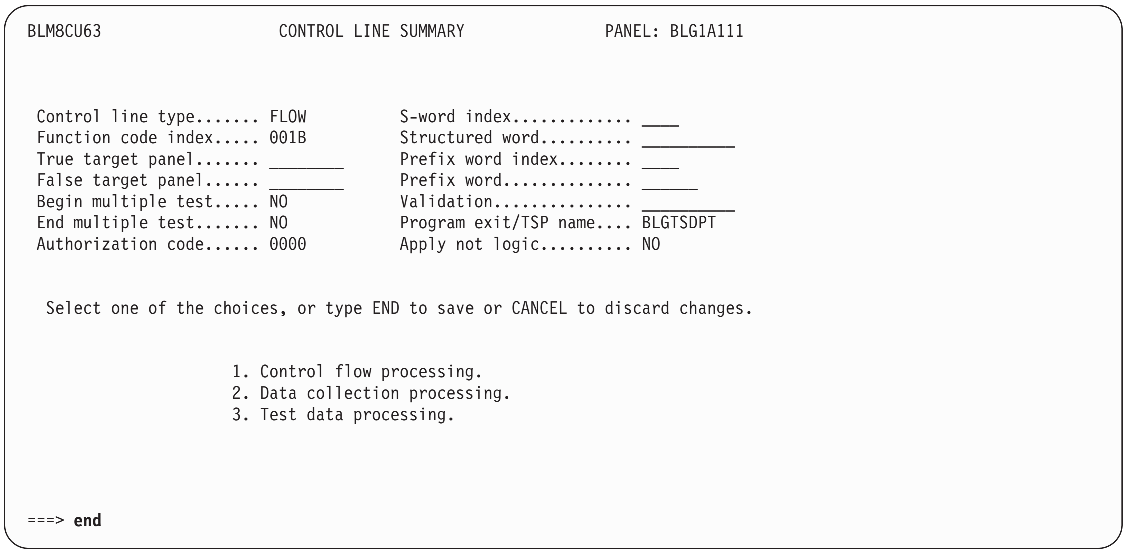
```
BLM8CU6B              DATA COLLECTION PROCESSING            PANEL: BLM1A111

 Enter 'add word' control data; cursor placement or input line entry allowed.

 1. Structured word index.... ____     Structured word.... __________
 2. Prefix index............ ____     Word acronym....... _________________
 3. Journal reply........... NO_      Prefix word........ ______
 4. Journal sequence......... _____    Validation......... _________________
 5. Cognize response......... NO_
 6. Cognize only p-word...... NO_
 7. Replace previous reply... YES
 8. Data is a date........... ___
 9. Cognize in mixed case?... NO_

      When you finish, type END to save or CANCEL to discard any changes.






===> 7,no
```

Type **end** and press Enter.



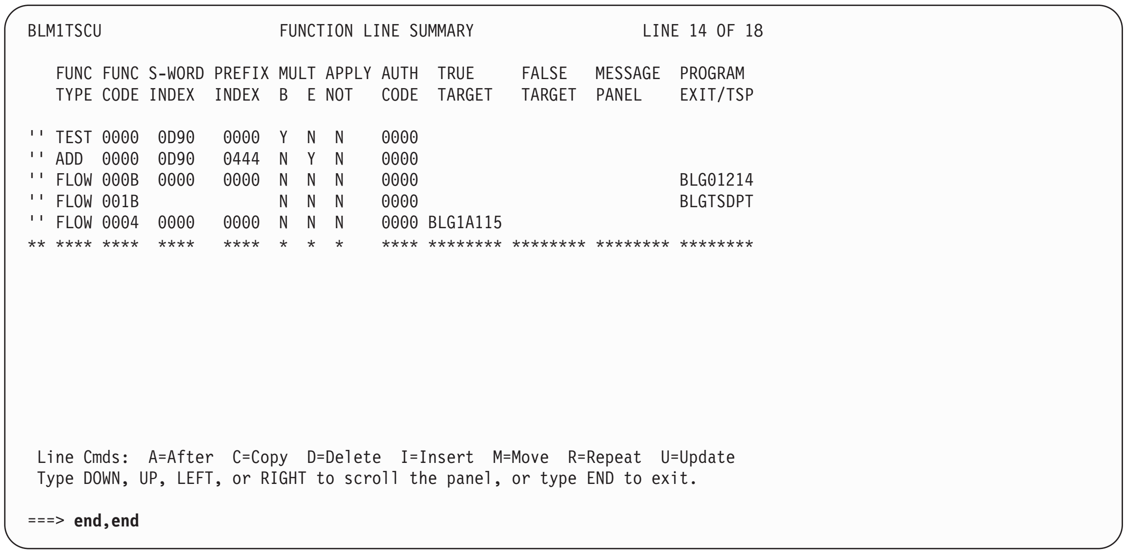
```
BLM8CU6B              DATA COLLECTION PROCESSING            PANEL: BLM1A111

 Enter 'add word' control data; cursor placement or input line entry allowed.

 1. Structured word index.... ____     Structured word.... __________
 2. Prefix index............ ____     Word acronym....... _________________
 3. Journal reply........... NO_      Prefix word........ ______
 4. Journal sequence......... _____    Validation......... _________________
 5. Cognize response......... NO_
 6. Cognize only p-word...... NO_
 7. Replace previous reply... NO
 8. Data is a date........... ___
 9. Cognize in mixed case?... NO_

      When you finish, type END to save or CANCEL to discard any changes.






===> end
```

Type **end** and press Enter to save the changes.



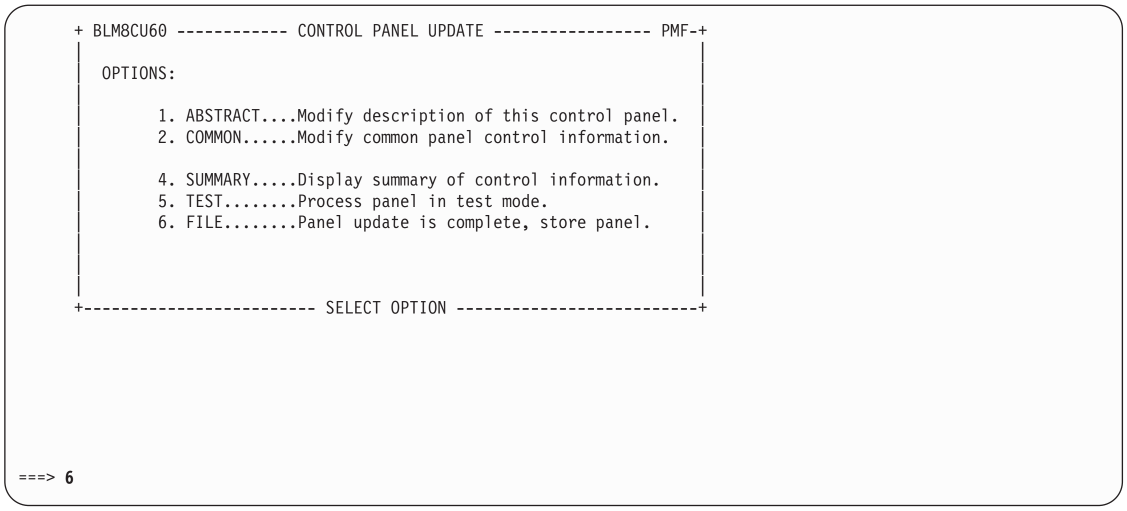
```
BLM8CU63                  CONTROL LINE SUMMARY              PANEL: BLG1A111



 Control line type....... FLOW         S-word index............. ____
 Function code index..... 001B         Structured word.......... __________
 True target panel....... ________     Prefix word index........ ____
 False target panel...... ________     Prefix word.............. ______
 Begin multiple test..... NO           Validation............... __________
 End multiple test....... NO           Program exit/TSP name.... BLGTSDPT
 Authorization code...... 0000         Apply not logic.......... NO


  Select one of the choices, or type END to save or CANCEL to discard changes.


                     1. Control flow processing.
                     2. Data collection processing.
                     3. Test data processing.




===> end
```

Type **end,end** and press Enter to return to the Control Panel Update panel.



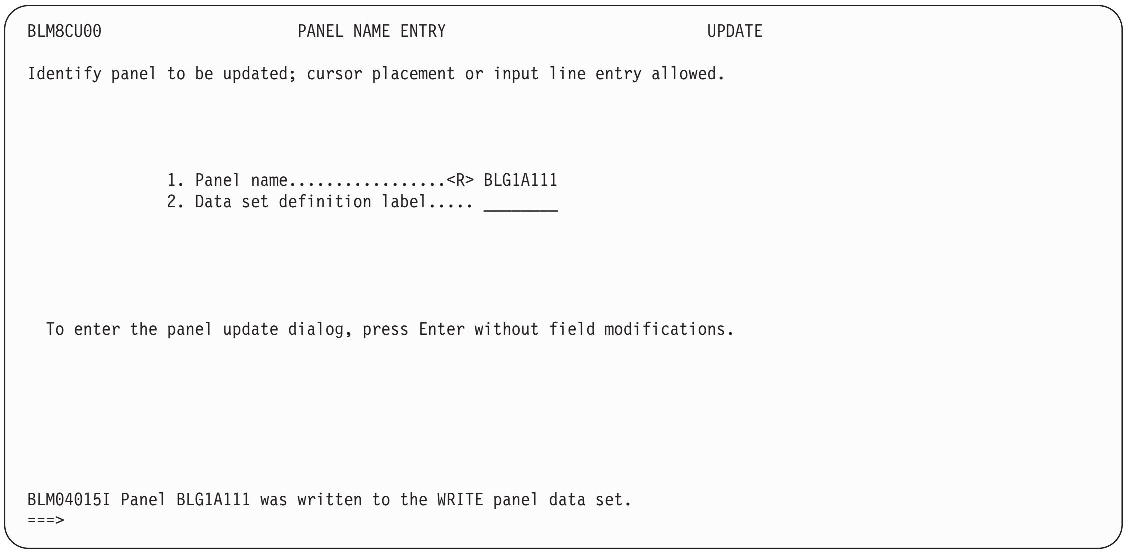
```
BLM1TSCU                  FUNCTION LINE SUMMARY                 LINE 14 OF 18

   FUNC FUNC S-WORD PREFIX MULT APPLY AUTH  TRUE     FALSE   MESSAGE  PROGRAM
   TYPE CODE INDEX  INDEX  B  E NOT   CODE  TARGET   TARGET  PANEL    EXIT/TSP

'' TEST 0000  0D90   0000  Y  N  N    0000
'' ADD  0000  0D90   0444  N  Y  N    0000
'' FLOW 000B  0000   0000  N  N  N    0000                            BLG01214
'' FLOW 001B               N  N  N    0000                            BLGTSDPT
'' FLOW 0004  0000   0000  N  N  N    0000 BLG1A115
** **** ****  ****   ****  *  *  *    **** ******** ******** ******** ********








 Line Cmds:  A=After  C=Copy  D=Delete  I=Insert  M=Move  R=Repeat  U=Update
 Type DOWN, UP, LEFT, or RIGHT to scroll the panel, or type END to exit.

===> end,end
```

Type **6** and press Enter to file the modified panel into the write panel data set.



```
      + BLM8CU60 ------------ CONTROL PANEL UPDATE ----------------- PMF-+
      |                                                                  |
      |  OPTIONS:                                                        |
      |                                                                  |
      |        1. ABSTRACT....Modify description of this control panel.  |
      |        2. COMMON......Modify common panel control information.   |
      |                                                                  |
      |        4. SUMMARY.....Display summary of control information.    |
      |        5. TEST........Process panel in test mode.                |
      |        6. FILE........Panel update is complete, store panel.     |
      |                                                                  |
      |                                                                  |
      |                                                                  |
      +------------------------- SELECT OPTION --------------------------+







===> 6
```

You will receive a message confirming that the panel was written to the write panel data set. Copy the panel to the read panel data set when you are ready for those changes to go into effect.

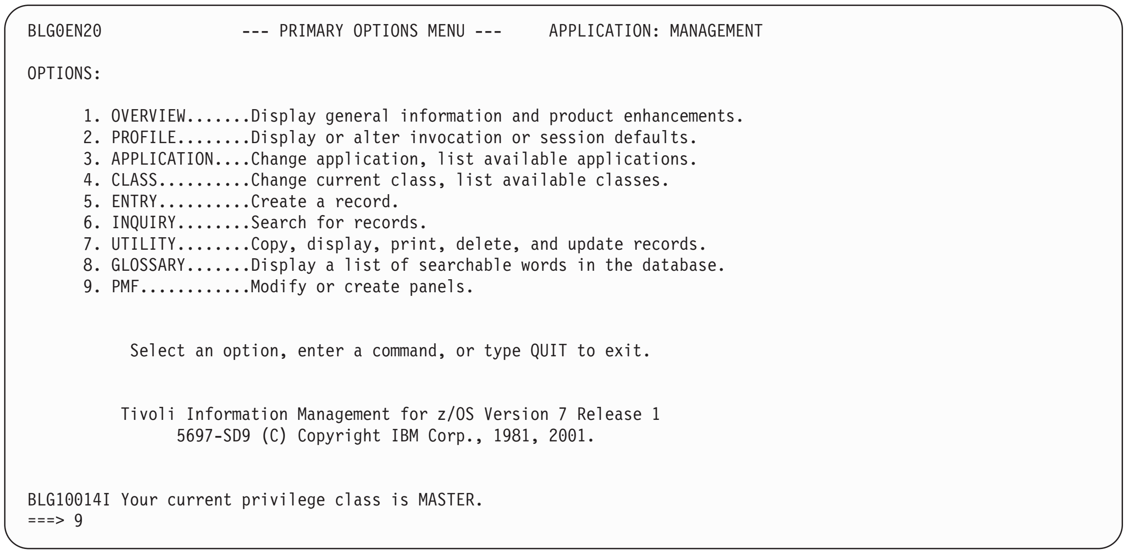


```
BLM8CU00                     PANEL NAME ENTRY                            UPDATE

Identify panel to be updated; cursor placement or input line entry allowed.




               1. Panel name................<R> BLG1A111
               2. Data set definition label..... ________





  To enter the panel update dialog, press Enter without field modifications.








BLM04015I Panel BLG1A111 was written to the WRITE panel data set.
===>
```

## Copying Panel BLG1A11Z

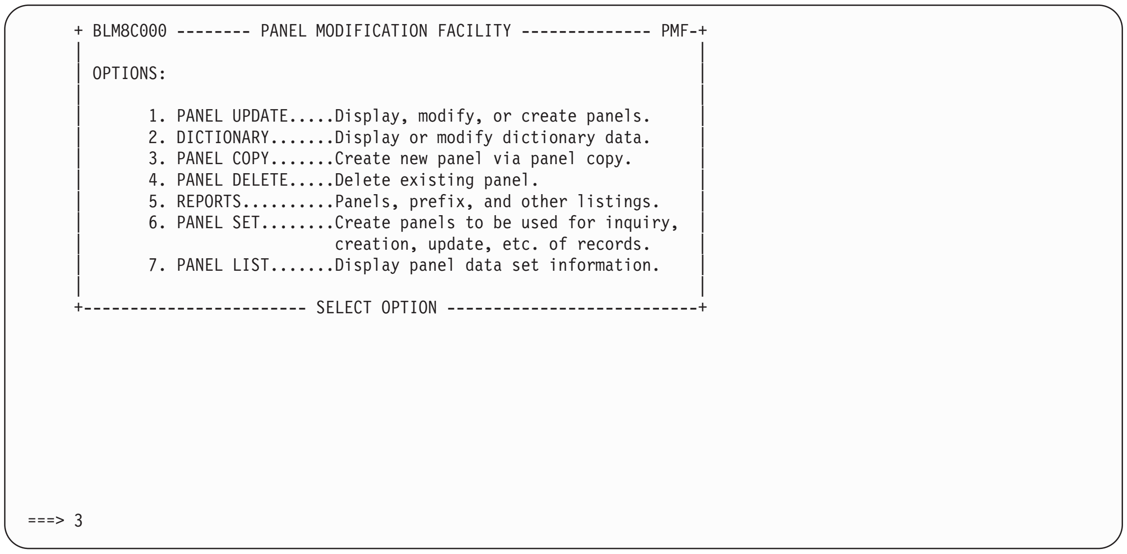
Use this procedure only if you have not previously modified panel BLG1A111 and therefore can replace it. Panel BLG1A11Z, as shipped from Tivoli, provides the appropriate function. The following process describes a means of copying BLG1A11Z from the base panel data set into the read panel data set and renaming it to BLG1A111.

From BLG0EN20, the Primary Options Menu, type **9** and press Enter.



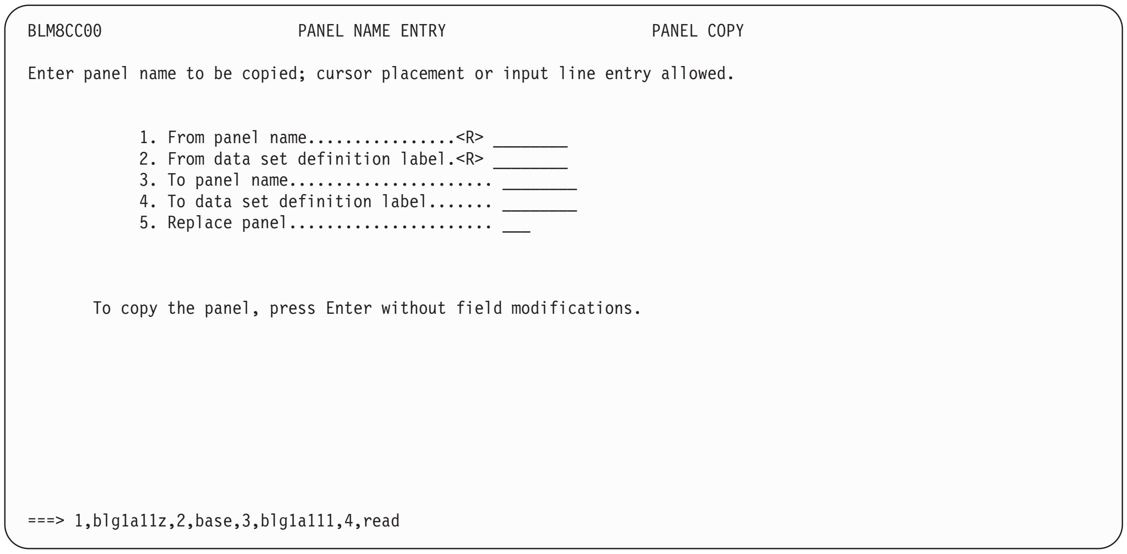
```
BLG0EN20              --- PRIMARY OPTIONS MENU ---     APPLICATION: MANAGEMENT

OPTIONS:

      1. OVERVIEW.......Display general information and product enhancements.
      2. PROFILE........Display or alter invocation or session defaults.
      3. APPLICATION....Change application, list available applications.
      4. CLASS..........Change current class, list available classes.
      5. ENTRY..........Create a record.
      6. INQUIRY........Search for records.
      7. UTILITY........Copy, display, print, delete, and update records.
      8. GLOSSARY.......Display a list of searchable words in the database.
      9. PMF............Modify or create panels.


           Select an option, enter a command, or type QUIT to exit.


          Tivoli Information Management for z/OS Version 7 Release 1
                5697-SD9 (C) Copyright IBM Corp., 1981, 2001.


BLG10014I Your current privilege class is MASTER.
===> 9
```

Type **3** and press Enter to copy a panel.



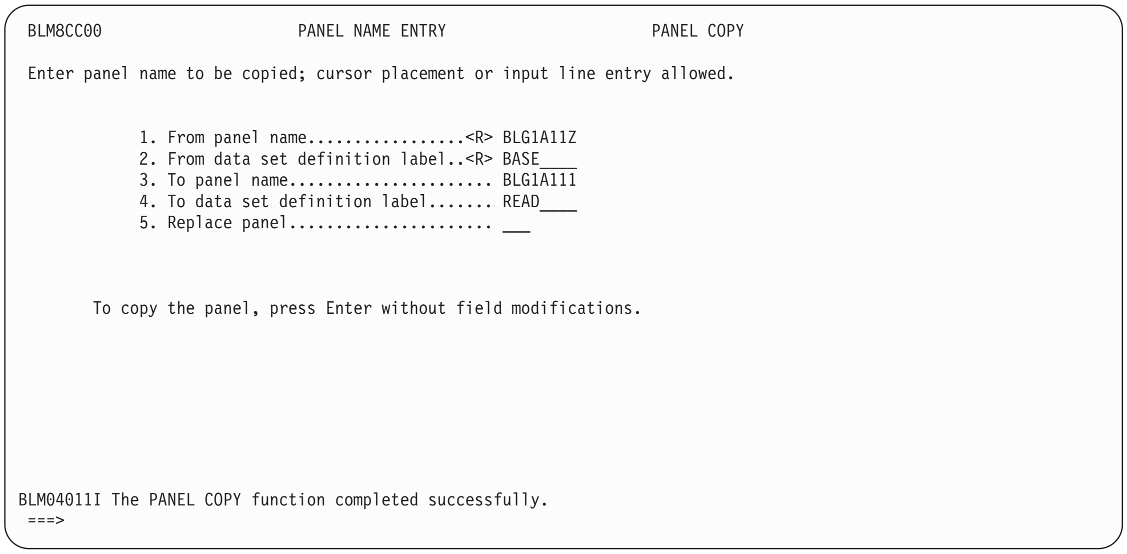
```
       + BLM8C000 -------- PANEL MODIFICATION FACILITY -------------- PMF-+
       |                                                                  |
       | OPTIONS:                                                         |
       |                                                                  |
       |       1. PANEL UPDATE.....Display, modify, or create panels.     |
       |       2. DICTIONARY.......Display or modify dictionary data.     |
       |       3. PANEL COPY.......Create new panel via panel copy.       |
       |       4. PANEL DELETE.....Delete existing panel.                 |
       |       5. REPORTS..........Panels, prefix, and other listings.    |
       |       6. PANEL SET........Create panels to be used for inquiry,  |
       |                           creation, update, etc. of records.     |
       |       7. PANEL LIST.......Display panel data set information.    |
       |                                                                  |
       +------------------------ SELECT OPTION ---------------------------+








===> 3
```

Type **1,blg1a11z,2,base,3,blg1a111,4,read** and press Enter.



```
BLM8CC00                   PANEL NAME ENTRY                    PANEL COPY

Enter panel name to be copied; cursor placement or input line entry allowed.


            1. From panel name................<R> ________
            2. From data set definition label.<R> ________
            3. To panel name..................... ________
            4. To data set definition label....... ________
            5. Replace panel..................... ___



       To copy the panel, press Enter without field modifications.








===> 1,blg1a11z,2,base,3,blg1a111,4,read
```

Press Enter again to cause panel BLG1A11Z to be copied from the BASE panel data set to the READ panel data set. You will receive a message confirming that the panel was copied.

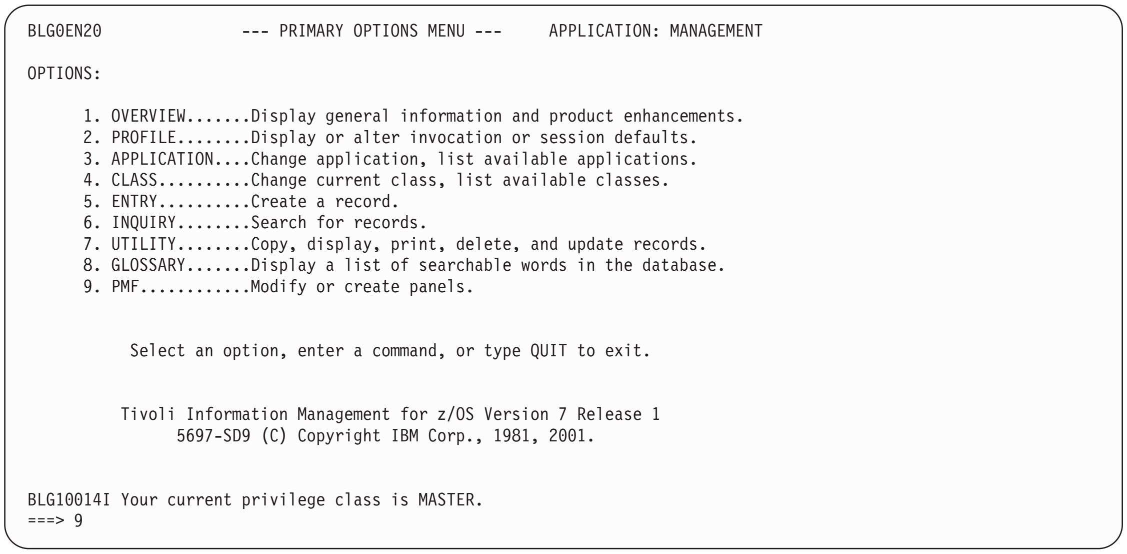


```
BLM8CC00                   PANEL NAME ENTRY                    PANEL COPY

Enter panel name to be copied; cursor placement or input line entry allowed.


            1. From panel name.................<R> BLG1A11Z
            2. From data set definition label..<R> BASE____
            3. To panel name..................... BLG1A111
            4. To data set definition label....... READ____
            5. Replace panel..................... ___



       To copy the panel, press Enter without field modifications.








BLM04011I The PANEL COPY function completed successfully.
 ===>
```

## Copying BLM1B04Z

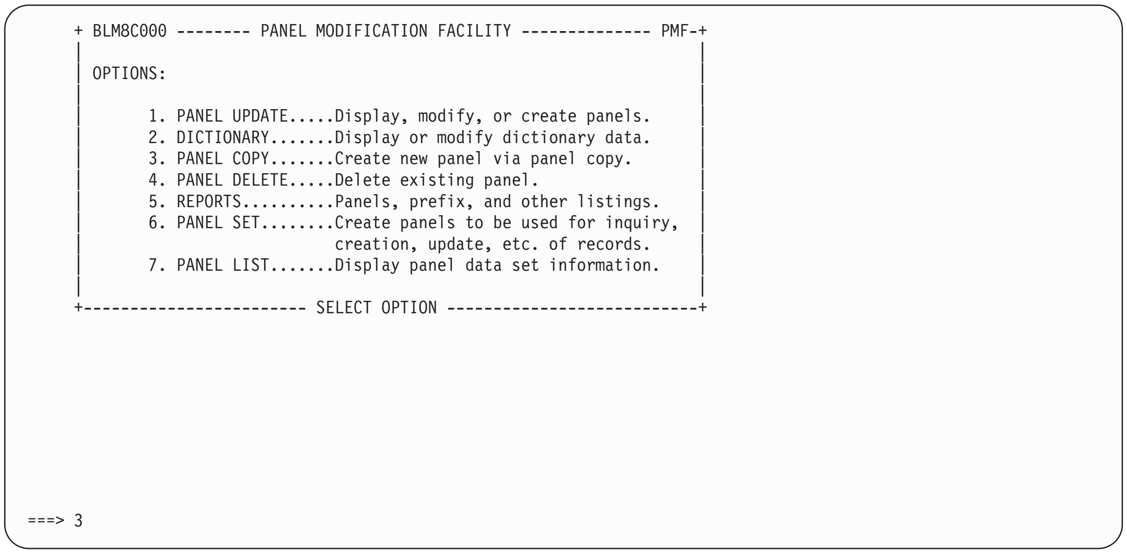
Use this procedure only if you have not previously modified panel BLM1B040 and can therefore replace it. Panel BLM1B04Z, as shipped from Tivoli, provides the appropriate function. The following process describes a means of copying BLM1B04Z from the base panel data set into the read panel data set and renaming it to BLM1B040.

From BLG0EN20, the Primary Options Menu, type **9** and press Enter.



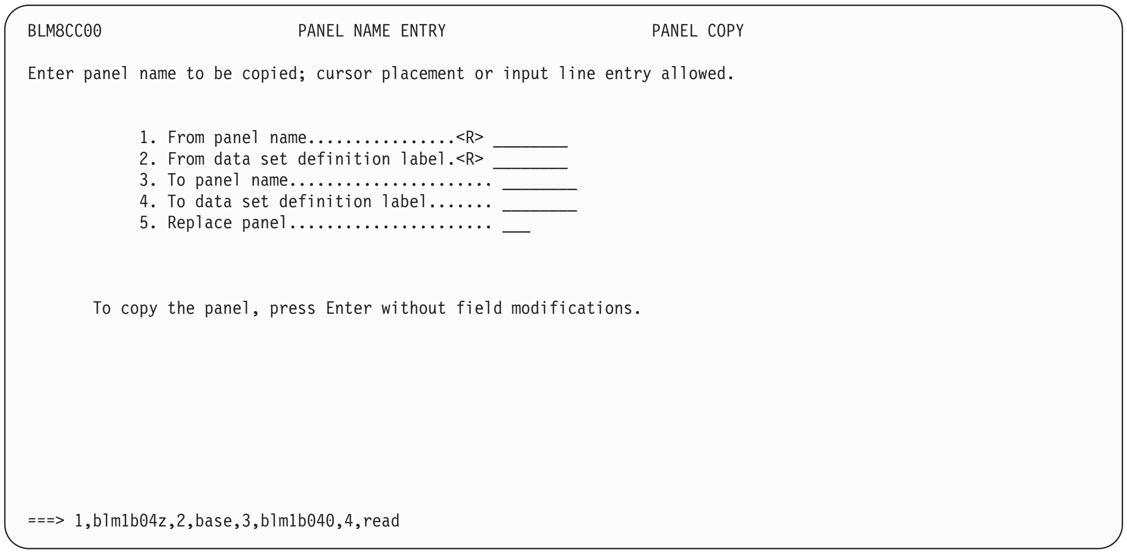
```
BLG0EN20              --- PRIMARY OPTIONS MENU ---     APPLICATION: MANAGEMENT

OPTIONS:

      1. OVERVIEW.......Display general information and product enhancements.
      2. PROFILE........Display or alter invocation or session defaults.
      3. APPLICATION....Change application, list available applications.
      4. CLASS..........Change current class, list available classes.
      5. ENTRY..........Create a record.
      6. INQUIRY........Search for records.
      7. UTILITY........Copy, display, print, delete, and update records.
      8. GLOSSARY.......Display a list of searchable words in the database.
      9. PMF............Modify or create panels.


           Select an option, enter a command, or type QUIT to exit.


          Tivoli Information Management for z/OS Version 7 Release 1
                5697-SD9 (C) Copyright IBM Corp., 1981, 2001.


BLG10014I Your current privilege class is MASTER.
===> 9
```

Type **3** and press Enter to copy a panel.



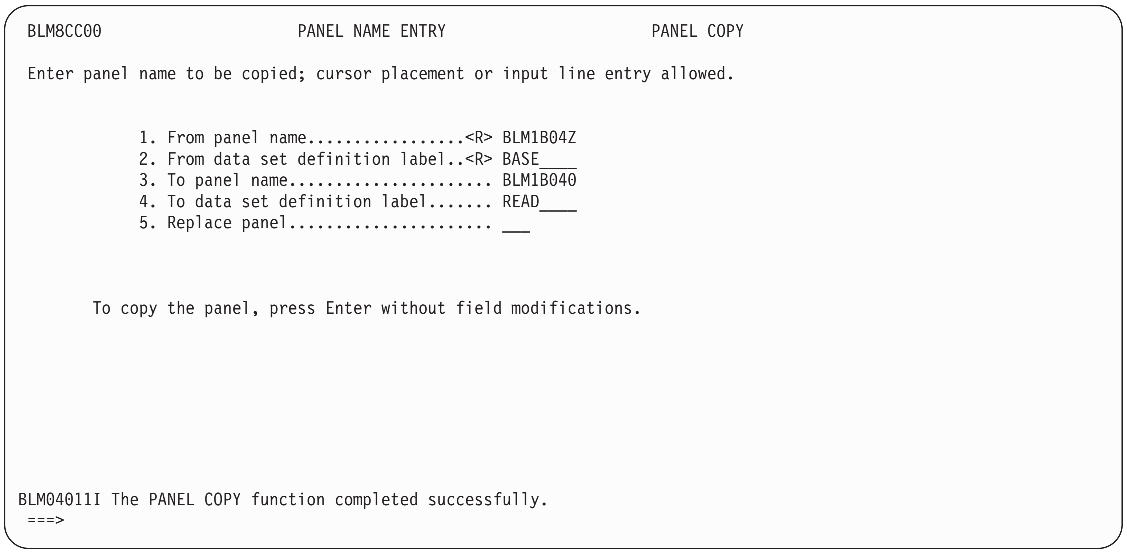
```
       + BLM8C000 -------- PANEL MODIFICATION FACILITY -------------- PMF-+
       |                                                                  |
       | OPTIONS:                                                         |
       |                                                                  |
       |       1. PANEL UPDATE.....Display, modify, or create panels.     |
       |       2. DICTIONARY.......Display or modify dictionary data.     |
       |       3. PANEL COPY.......Create new panel via panel copy.       |
       |       4. PANEL DELETE.....Delete existing panel.                 |
       |       5. REPORTS..........Panels, prefix, and other listings.    |
       |       6. PANEL SET........Create panels to be used for inquiry,  |
       |                           creation, update, etc. of records.     |
       |       7. PANEL LIST.......Display panel data set information.    |
       |                                                                  |
       +----------------------- SELECT OPTION ---------------------------+







===> 3
```

Type **1,blm1b04z,2,base,3,blm1b040,4,read** and press Enter.



```
BLM8CC00                    PANEL NAME ENTRY                    PANEL COPY

Enter panel name to be copied; cursor placement or input line entry allowed.


            1. From panel name................<R> ________
            2. From data set definition label.<R> ________
            3. To panel name..................... ________
            4. To data set definition label...... ________
            5. Replace panel..................... ___



       To copy the panel, press Enter without field modifications.









===> 1,blm1b04z,2,base,3,blm1b040,4,read
```

Press Enter again to cause panel BLM1B04Z to be copied from the BASE panel data set to the READ panel data set and renamed to BLM1B040. You will receive a message confirming that the panel was copied.



```
BLM8CC00                    PANEL NAME ENTRY                    PANEL COPY

Enter panel name to be copied; cursor placement or input line entry allowed.


            1. From panel name................<R> BLM1B04Z
            2. From data set definition label..<R> BASE____
            3. To panel name..................... BLM1B040
            4. To data set definition label...... READ____
            5. Replace panel..................... ___



       To copy the panel, press Enter without field modifications.









BLM04011I The PANEL COPY function completed successfully.
 ===>
```

## Starting the Notification Server

The notification server is a TSX, BLGTSDNS, that is run by a batch Information Management job; the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* contains information about starting Information Management as a batch job in a section entitled “Starting Tivoli Information Management for z/OS in Batch Mode”. Because this is intended to be a long-running job, specify the TIME parameter of the EXEC statement such that the batch job does not time out. The user ID that is associated with this batch job must have authority to display people records and problem records. Before you start the notification server, copy BLGTSDNS to your own TSX data set and update it according to your environment. See “Setting Up the Notification Server” on page 321. Also, specify your TSX data set in the BLGTSX DD statement in your batch job JCL. The following is an example of a statement that starts Information Management in batch mode and runs the notification server TSX BLGTSDNS. It uses session-parameters member BLGSES00.

```
ISPSTART PGM(BLGINIT) PARM('SESS(00) TSP(BLGTSDNS) IRC(QUIT)')
```

### Stopping the Notification Server

You can stop the notification server in several ways:

- By issuing an operator command to close the control RDR. Assuming that the control RDR is named BLGTSDC1, this is an example of an operator command to close it, where `sblx1` is the name of your BLX-SP:

  ```
  MODIFY sblx1,RDR,FLUSH=BLGTSDC1
  ```

- By running a TSX that closes the control RDR. Assuming that the control RDR is named BLGTSDC1, this is an example of a TSX control line that closes the RDR:

  ```
  CALL BLGTSX 'CLOSERRES','BLGTSDC1'
  ```

- By canceling the batch job that is running; you should do this only if necessary, because if you cancel the batch job, it does not go to normal completion and both the control RDR and the data RDR remain open.

## TSD Setup

Following are the steps that must occur on TSD before using the Tivoli Service Desk Bridge:

1. Ensure that the MRES running on Information Management was started with pre-started API sessions (see step 1 on page 319).
2. Start the HLAPI/NT requester on the TSD application server, if you have not already done so.
3. Set up the mapper; information about the mapping program can be found in the *Tivoli Service Desk Networking Guide*. You might want to build an alias table to be used by the mapper. A description of how to use the Table Build Utility to build an alias table can be found in the *Tivoli Information Management for z/OS Application Program Interface Guide*.
4. Set up the TSD listener program error processing. Two sample TSXs, BLGTSDER and BLGTSDE1, that process errors originating from the listener program, are shipped by Tivoli Information Management for z/OS in the SBLMTSX data set. These TSXs are described in "Setting Up Error Processing for the TSD Listener Program".
5. Start the TSD listener program; the procedure for doing this is described in the *Tivoli Service Desk Networking Guide*.
6. Ensure that people records have been created for any TSD users to whom Information Management will assign problems. The method of creating people records created for TSD users is contained in the *Tivoli Service Desk Networking Guide*.

### Setting Up Error Processing for the TSD Listener Program

Tivoli Information Management for z/OS provides two sample TSXs, BLGTSDER and BLGTSDE1.

BLGTSDER is a sample TSX to process errors coming from the TSD listener program. Decide how these errors will be handled at your installation. BLGTSDER shows how to create a Information Management problem record and a TEC event. BLGTSDER uses the Information Management TEC Event Adapter TSX BLGTAGSD to create TEC events. If you want to create TEC events when errors are detected by the TSD Listener program, TSX

BLGTAGSD must be running. See step 3 on page 397 for information on the BLGTECAD JCL that runs the Information Management TEC Event Adapter as a batch job.

**Note:** If you are going to create Tivoli Enterprise Console (TEC) events to handle Tivoli Service Desk Bridge errors, you must install Tivoli Information Management for z/OS TEC Event Adapter.

BLGTSDER can also call BLGTSDE1, which can send an e-mail message when an error is detected by the TSD listener program.

BLGTSDER and BLGTSDE1 are shipped in the SBLMTSX data set. Copy them to your TSX data set and update them as appropriate. Specify your TSX data set on the BLGTSX DD statement in your MRES procedure.

# 28 Using the Tivoli Service Desk Bridge

The Tivoli Service Desk Bridge permits records to be transferred from a user on one system to a user on another system. Additionally, when a record has been transferred from Tivoli Information Management for z/OS to TSD, an Information Management user can request that the record in the Information Management database be updated with current information. An Information Management user can also request that the record be transferred back to Information Management. The Tivoli Service Desk Bridge also provides the ability to send the solution for a closed Information Management problem record to the TSD knowledge database.

**Note:** When one of the functions described in this chapter is being used, the record in the Information Management database is locked until the processing is complete. The record locking is automatic, with no external interface. The TSD Bridge Cleanup function is available to the database administrator and can be used if the record seems to be permanently locked. The *Tivoli Information Management for z/OS Program Administration Guide and Reference* contains more information about the TSD Bridge Cleanup function.

## Transferring a Problem from Information Management to TSD

Information Management people records with a **Person role** of TSDUSER or TSDGROUP represent TSD users. These records are created in the Information Management database by the TSD system. In TSDUSER and TSDGROUP people records, there is also a TSD user ID. When a problem record is assigned to a TSD user ID that is specified in a TSDUSER or TSDGROUP people record, ownership for the problem record is transferred to TSD.

To transfer a problem record to TSD, assign the record to a TSD help desk analyst. To assign the record, specify the TSD user ID for a TSDUSER or TSDGROUP people record in the **Assignee name** field of a problem record.

In the following scenario, Frank is a TSD user. There is a people record for Frank with a Person role of TSDUSER and his TSD user ID is **frank1**. Problem record 131 is assigned to Ashley, an Information Management user. Problem record 131 is to be transferred to Frank. From a Tivoli Information Management for z/OS panel, type **upd r 131** and press Enter to update the record.

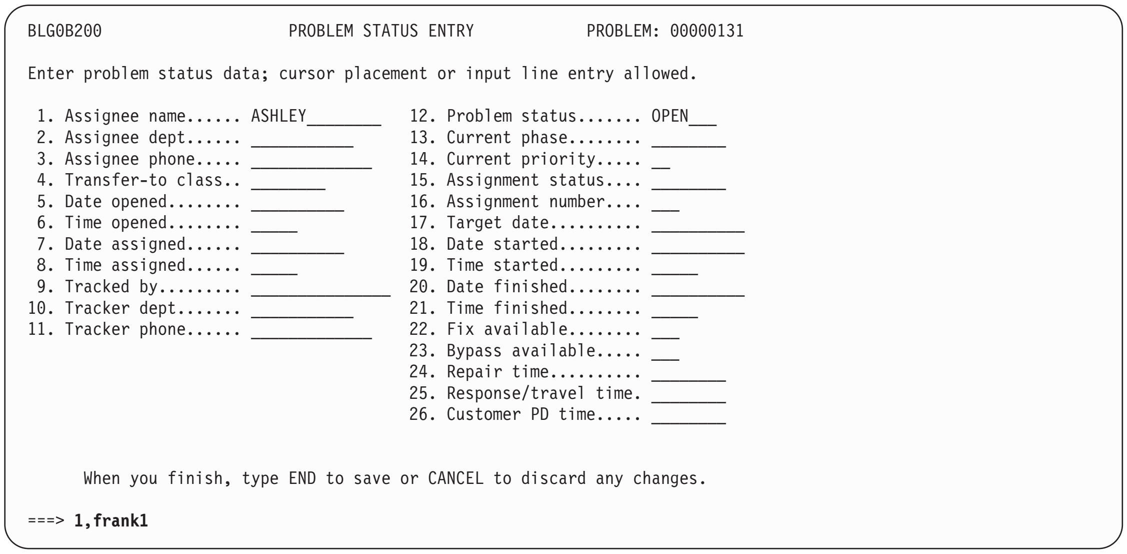
```
BLG0EN20              --- PRIMARY OPTIONS MENU ---     APPLICATION: MANAGEMENT

OPTIONS:

      1. OVERVIEW.......Display general information and product enhancements.
      2. PROFILE........Display or alter invocation or session defaults.
      3. APPLICATION....Change application, list available applications.
      4. CLASS..........Change current class, list available classes.
      5. ENTRY..........Create a record.
      6. INQUIRY........Search for records.
      7. UTILITY........Copy, display, print, delete, and update records.
      8. GLOSSARY.......Display a list of searchable words in the database.
      9. PMF............Modify or create panels.


           Select an option, enter a command, or type QUIT to exit.


          Tivoli Information Management for z/OS Version 7 Release 1
               5697-SD9 (C) Copyright IBM Corp., 1981, 2001.

===> upd r 131
```

Type **2** and press Enter to specify Status data.

```
BLG0BU00                    PROBLEM SUMMARY               PROBLEM: 00000131

Reported by............ CHARLES          Problem status.......... OPEN
Assignee name.......... ASHLEY           Current phase........... ________
Tracked by............. _______________  Current priority........ __
Network name........... ________         Owning priv. class...... ________
System name............ ________         Entry priv. class....... MASTER
Program name........... ________         Date entered............ 05/25/1999
Device name............ ________         Time entered............ 11:07
Key item affected...... ________         Date last altered....... 05/25/1999

Description............ sample problem


 Select one of the following, type END to save your changes, or type CANCEL
 to discard your changes.
              1. Reporter data.          6. Supplemental data.
              2. Status data.            7. Synopsis data.
              3. Close data.             8. Freeform text.
              4. Symptom data.           9. File record.
              5. Resolution data.       10. Create solution and file record.


===> 2
```

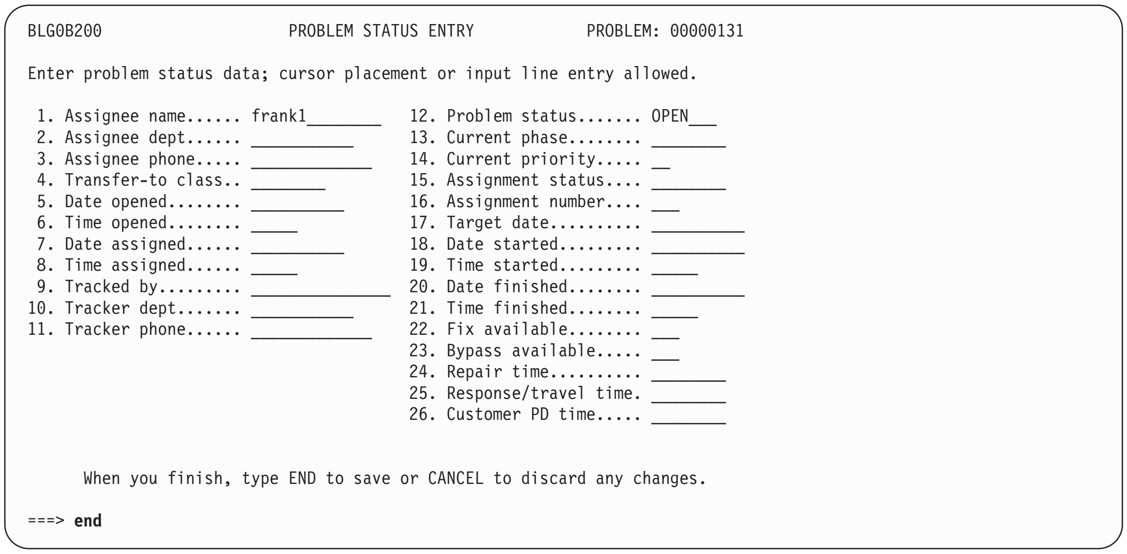
The Status Entry panel, BLG0B200, indicates that the record is assigned to Ashley.

To transfer the record to Frank, enter Frank's TSD user ID `frank1` into the **Assignee name** by typing **1,frank1** and pressing Enter.



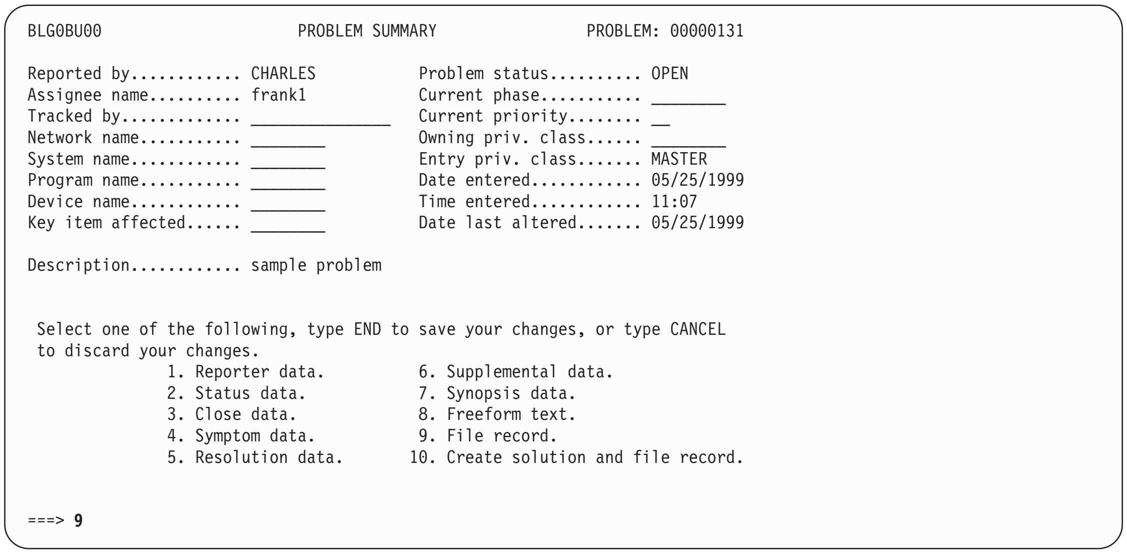
```
BLG0B200                  PROBLEM STATUS ENTRY           PROBLEM: 00000131

Enter problem status data; cursor placement or input line entry allowed.

 1. Assignee name...... ASHLEY________   12. Problem status....... OPEN___
 2. Assignee dept...... ___________      13. Current phase........ ________
 3. Assignee phone..... _____________    14. Current priority..... __
 4. Transfer-to class.. ________         15. Assignment status.... ________
 5. Date opened........ __________       16. Assignment number.... ___
 6. Time opened........ _____            17. Target date.......... __________
 7. Date assigned...... __________       18. Date started......... __________
 8. Time assigned...... _____            19. Time started......... _____
 9. Tracked by......... _______________  20. Date finished........ __________
10. Tracker dept....... ___________      21. Time finished........ _____
11. Tracker phone...... _____________    22. Fix available........ ___
                                         23. Bypass available..... ___
                                         24. Repair time.......... ________
                                         25. Response/travel time. ________
                                         26. Customer PD time..... ________


      When you finish, type END to save or CANCEL to discard any changes.

===> 1,frank1
```

Then type **end** on the command line and press Enter to return to the Problem Summary panel.



```
BLG0B200                  PROBLEM STATUS ENTRY           PROBLEM: 00000131

Enter problem status data; cursor placement or input line entry allowed.

 1. Assignee name...... frank1________   12. Problem status....... OPEN___
 2. Assignee dept...... ___________      13. Current phase........ ________
 3. Assignee phone..... _____________    14. Current priority..... __
 4. Transfer-to class.. ________         15. Assignment status.... ________
 5. Date opened........ __________       16. Assignment number.... ___
 6. Time opened........ _____            17. Target date.......... __________
 7. Date assigned...... __________       18. Date started......... __________
 8. Time assigned...... _____            19. Time started......... _____
 9. Tracked by......... _______________  20. Date finished........ __________
10. Tracker dept....... ___________      21. Time finished........ _____
11. Tracker phone...... _____________    22. Fix available........ ___
                                         23. Bypass available..... ___
                                         24. Repair time.......... ________
                                         25. Response/travel time. ________
                                         26. Customer PD time..... ________


      When you finish, type END to save or CANCEL to discard any changes.

===> end
```

Type **9** and press Enter to file the record.



```
BLG0BU00                   PROBLEM SUMMARY               PROBLEM: 00000131

Reported by........... CHARLES          Problem status.......... OPEN
Assignee name......... frank1           Current phase........... ________
Tracked by............ _______________  Current priority........ __
Network name.......... ________         Owning priv. class...... ________
System name........... ________         Entry priv. class....... MASTER
Program name.......... ________         Date entered............ 05/25/1999
Device name........... ________         Time entered............ 11:07
Key item affected..... ________         Date last altered....... 05/25/1999

Description........... sample problem


 Select one of the following, type END to save your changes, or type CANCEL
 to discard your changes.
              1. Reporter data.          6. Supplemental data.
              2. Status data.            7. Synopsis data.
              3. Close data.             8. Freeform text.
              4. Symptom data.           9. File record.
              5. Resolution data.       10. Create solution and file record.


===> 9
```

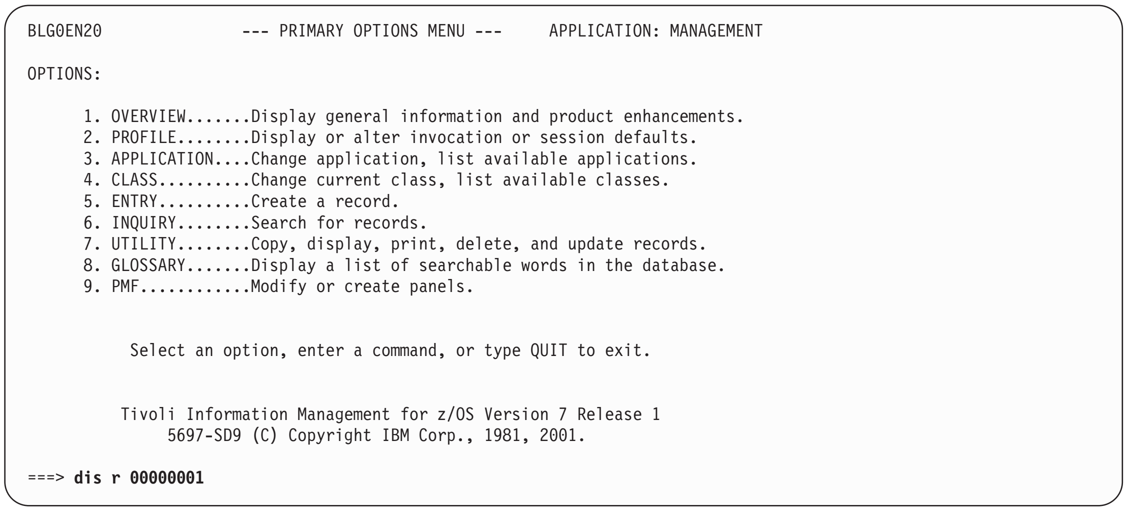
When the problem record is filed in the Information Management database, the Tivoli Service Desk Bridge determines that the Assignee name matches the TSD user ID of a TSD user (a user whose **Person role** is either **TSDUSER** or **TSDGROUP**). It then initiates the process to transfer the problem to TSD by adding notification data to the record, and, if the notification server is active, places the notification data on the data RDR.

When the notification data is added to the problem record, the problem record is locked from further updates. When the transfer is complete, the record continues to be locked; however, in this instance, it is locked because ownership of the record now belongs to TSD. An Information Management user can request that the copy of the record in the Information Management database be refreshed with current information from TSD and can also request that the problem be transferred back to Information Management.

## Resume Ownership

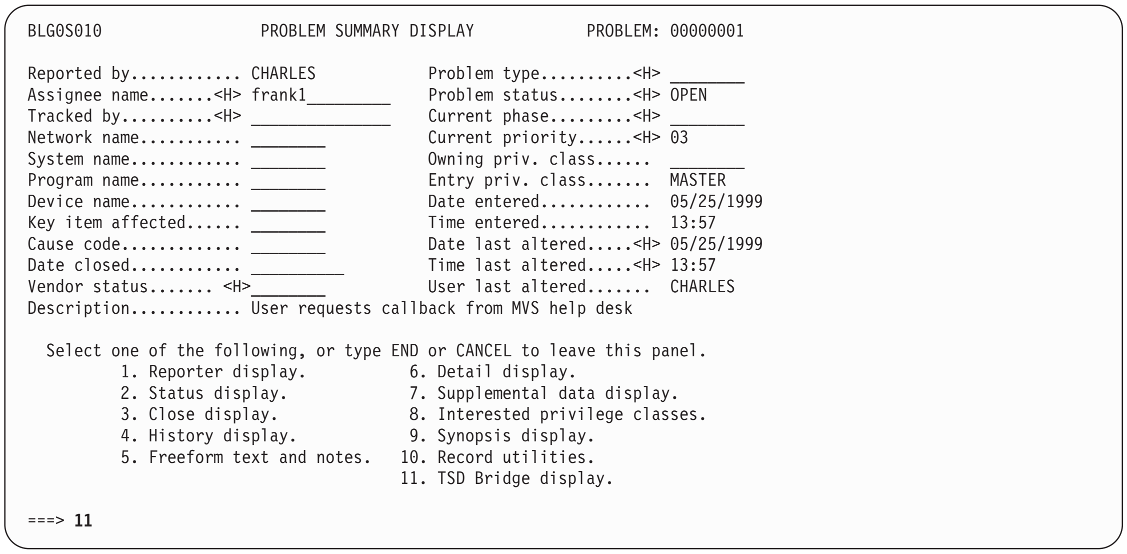
As described in “Tivoli Service Desk Bridge Overview” on page 315, a problem record can be owned by Information Management (in which case any Information Management user with appropriate authority can access it) or it is owned by TSD (in which case only the owner of a problem record can add information or update the record). If an Information Management analyst wishes to update an Information Management problem record that has been transferred to TSD, the analyst must resume ownership of the record. Because TSD assigns the record to this user when it returns the record to Information Management, TSD must have knowledge of the Information Management user. Therefore, the only Information Management user empowered to resume ownership of a record owned by a TSD user must have a **Person role** of **TSD390&TSD**. To resume ownership, the Information Management analyst must display the record and select **1. Resume ownership** from the TSD Bridge Display panel. If the analyst is a TSD390&TSD user, a request to resume ownership of the record will be sent to TSD.

In the following scenario, problem record 00000001 has been transferred to TSD. It is currently assigned to Frank (who has a TSD user ID of frank1). Ashley is an Information Management user who wants to resume ownership of the problem. Ashley’s MVS user ID is ASHLEY. Ashley has a people record; its record ID matches her MVS user ID. The ASHLEY people record indicates that she has a Person role of TSD390&TSD. This allows her to resume ownership. From a Tivoli Information Management for z/OS panel, type **dis r 00000001** and press Enter to display the record.



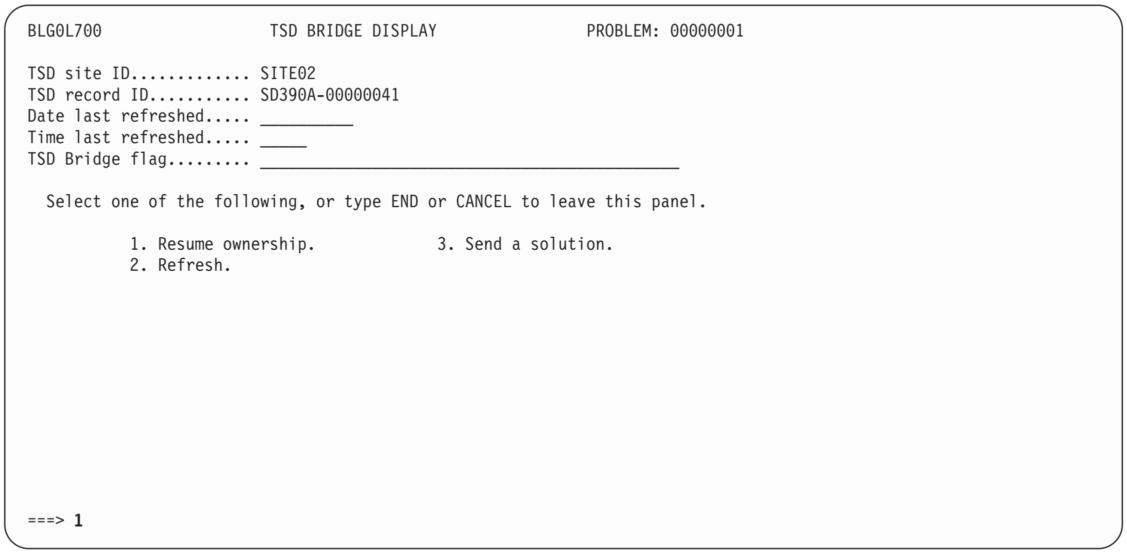
```
BLG0EN20              --- PRIMARY OPTIONS MENU ---     APPLICATION: MANAGEMENT

OPTIONS:

      1. OVERVIEW.......Display general information and product enhancements.
      2. PROFILE........Display or alter invocation or session defaults.
      3. APPLICATION....Change application, list available applications.
      4. CLASS..........Change current class, list available classes.
      5. ENTRY..........Create a record.
      6. INQUIRY........Search for records.
      7. UTILITY........Copy, display, print, delete, and update records.
      8. GLOSSARY.......Display a list of searchable words in the database.
      9. PMF............Modify or create panels.


           Select an option, enter a command, or type QUIT to exit.


          Tivoli Information Management for z/OS Version 7 Release 1
               5697-SD9 (C) Copyright IBM Corp., 1981, 2001.

===> dis r 00000001
```

Type **11** and press Enter to select TSD Bridge display.



```
BLG0S010                PROBLEM SUMMARY DISPLAY           PROBLEM: 00000001

Reported by............ CHARLES           Problem type..........<H> ________
Assignee name.......<H> frank1________    Problem status........<H> OPEN
Tracked by..........<H> _______________   Current phase.........<H> ________
Network name........... ________          Current priority......<H> 03
System name............ ________          Owning priv. class......  ________
Program name........... ________          Entry priv. class.......  MASTER
Device name............ ________          Date entered............  05/25/1999
Key item affected...... ________          Time entered............  13:57
Cause code............. ________          Date last altered.....<H> 05/25/1999
Date closed............ __________        Time last altered.....<H> 13:57
Vendor status....... <H>________          User last altered.......  CHARLES
Description............ User requests callback from MVS help desk

  Select one of the following, or type END or CANCEL to leave this panel.
          1. Reporter display.           6. Detail display.
          2. Status display.             7. Supplemental data display.
          3. Close display.              8. Interested privilege classes.
          4. History display.            9. Synopsis display.
          5. Freeform text and notes.   10. Record utilities.
                                        11. TSD Bridge display.

===> 11
```

Type **1** and press Enter to resume ownership.



```
BLG0L700                 TSD BRIDGE DISPLAY               PROBLEM: 00000001

TSD site ID............ SITE02
TSD record ID.......... SD390A-00000041
Date last refreshed..... __________
Time last refreshed..... _____
TSD Bridge flag......... _____________________________________________

  Select one of the following, or type END or CANCEL to leave this panel.

           1. Resume ownership.            3. Send a solution.
           2. Refresh.









===> 1
```

If the user ID of the analyst requesting resume ownership matches that record ID of a people record that has a Person role of TSD390&TSD, a request to resume ownership is sent to TSD. While the request is being processed, the user can work with other records. When the request has completed, the user receives notification that ownership has been returned. Additionally, the record will be assigned to that user.

**Note:** The fields on BLG0L700 are as follows:

**TSD site ID**
If this field is blank or contains a value of LOCAL, the record is owned by Information Management; otherwise it is owned by TSD.

**TSD record ID**
If this field contains a value, then a copy of this problem exists in the TSD database.

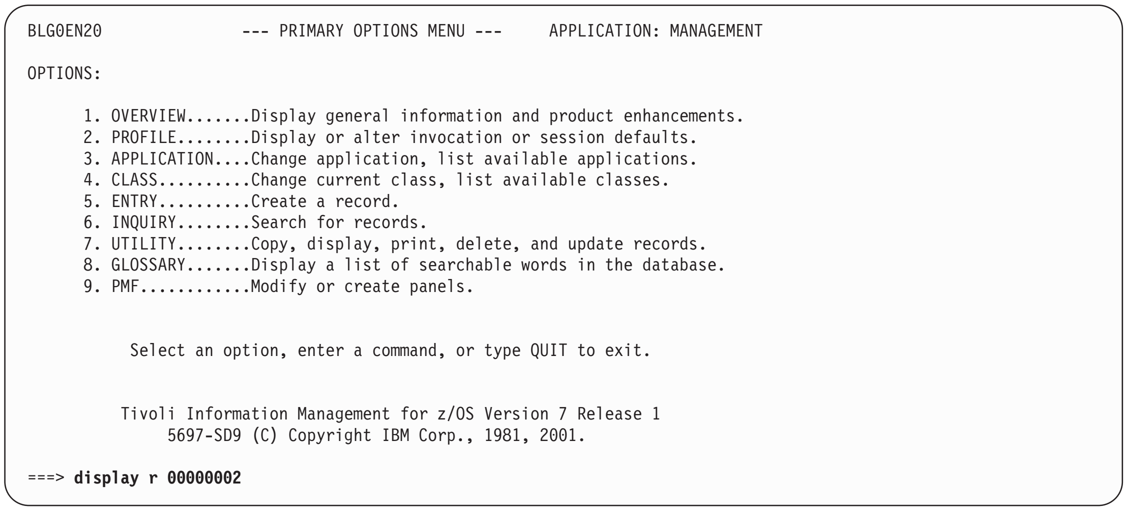
**Date last refreshed**
The date that this record was last refreshed by TSD.

**Time last refreshed**
The time that this record was last refreshed by TSD.

**TSD Bridge flag**
If this field contains a value, a transaction is in the process of being sent to TSD for this record.

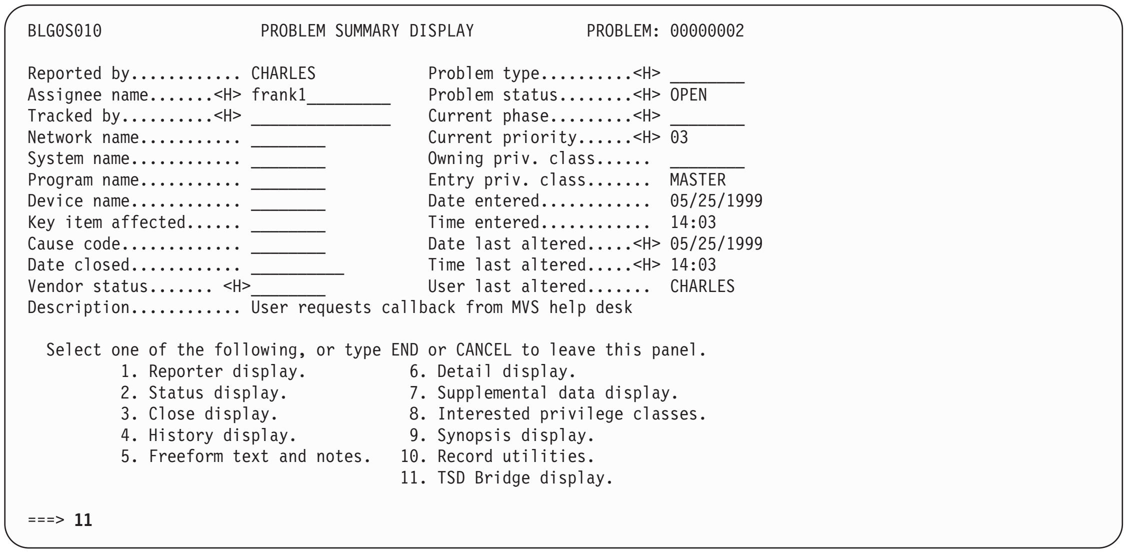
## Refresh

If a record has been transferred from Information Management to TSD, the Information Management analyst can use the Refresh function to have the Information Management record updated with the current information from the TSD system. To refresh the record, the Information Management analyst must display the record and select **2. Refresh** from the TSD Bridge Display panel. From a Tivoli Information Management for z/OS panel, type **display r 00000002** and press Enter to display the record.



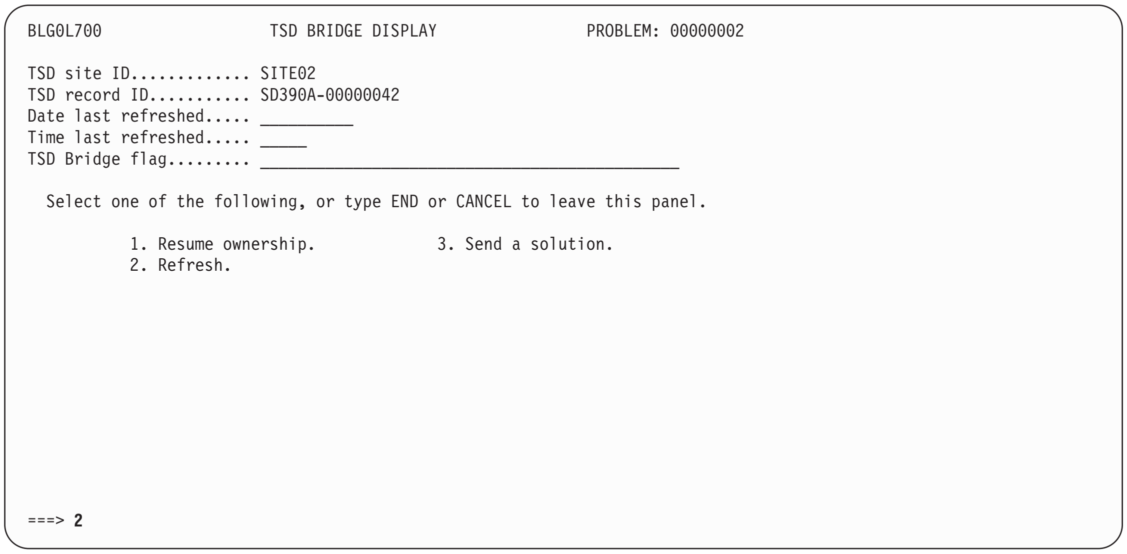
```
BLG0EN20              --- PRIMARY OPTIONS MENU ---     APPLICATION: MANAGEMENT

OPTIONS:

      1. OVERVIEW.......Display general information and product enhancements.
      2. PROFILE........Display or alter invocation or session defaults.
      3. APPLICATION....Change application, list available applications.
      4. CLASS..........Change current class, list available classes.
      5. ENTRY..........Create a record.
      6. INQUIRY........Search for records.
      7. UTILITY........Copy, display, print, delete, and update records.
      8. GLOSSARY.......Display a list of searchable words in the database.
      9. PMF............Modify or create panels.


           Select an option, enter a command, or type QUIT to exit.


          Tivoli Information Management for z/OS Version 7 Release 1
               5697-SD9 (C) Copyright IBM Corp., 1981, 2001.

===> display r 00000002
```

Type **11** and press Enter to select TSD Bridge display.



```
BLG0S010                PROBLEM SUMMARY DISPLAY           PROBLEM: 00000002

Reported by............ CHARLES           Problem type..........<H> ________
Assignee name.......<H> frank1________    Problem status........<H> OPEN
Tracked by..........<H> _______________   Current phase.........<H> ________
Network name........... ________          Current priority......<H> 03
System name............ ________          Owning priv. class......  ________
Program name........... ________          Entry priv. class.......  MASTER
Device name............ ________          Date entered............  05/25/1999
Key item affected...... ________          Time entered............  14:03
Cause code............. ________          Date last altered.....<H> 05/25/1999
Date closed............ __________        Time last altered.....<H> 14:03
Vendor status....... <H>________          User last altered.......  CHARLES
Description............ User requests callback from MVS help desk

  Select one of the following, or type END or CANCEL to leave this panel.
          1. Reporter display.          6. Detail display.
          2. Status display.            7. Supplemental data display.
          3. Close display.             8. Interested privilege classes.
          4. History display.           9. Synopsis display.
          5. Freeform text and notes.  10. Record utilities.
                                       11. TSD Bridge display.

===> 11
```

The fields **Date last refreshed** and **Time last refreshed** on the TSD Bridge Display panel indicate the last time the record was refreshed since transferred to TSD. Type **2** and press Enter to send a request to TSD to refresh the record. While the request to refresh the record is being processed by TSD, the user can work with other records. When the request has completed, the user receives notification that the record has been refreshed.



```
BLG0L700                 TSD BRIDGE DISPLAY               PROBLEM: 00000002

TSD site ID............ SITE02
TSD record ID.......... SD390A-00000042
Date last refreshed..... __________
Time last refreshed..... _____
TSD Bridge flag......... _____________________________________________

  Select one of the following, or type END or CANCEL to leave this panel.

           1. Resume ownership.             3. Send a solution.
           2. Refresh.

===> 2
```

**Note:** The contents of the fields on this panel are explained on page 358.

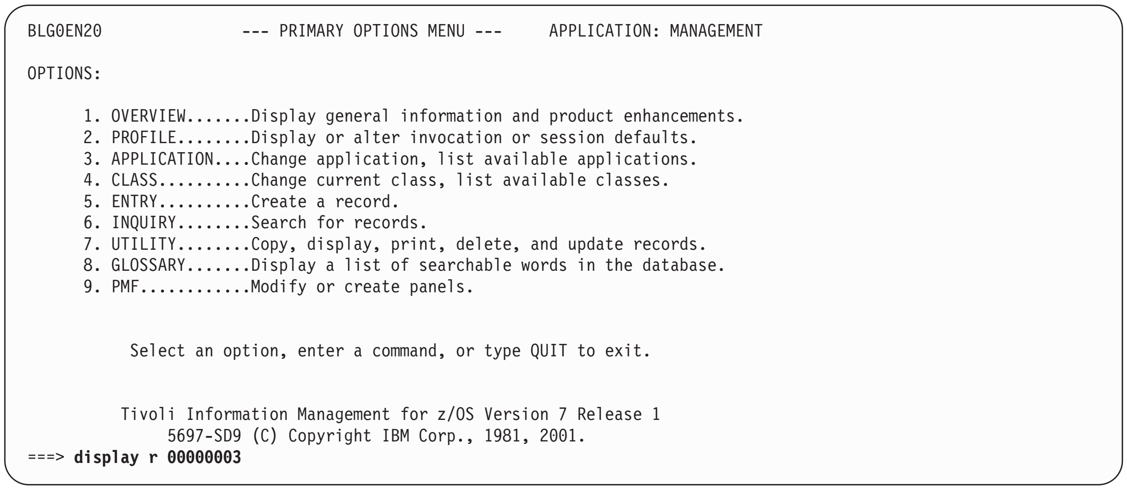
## Send a Solution

When an Information Management user resolves a problem, the Send–a–solution function can be used to transmit the resolution to the problem to the TSD knowledge database. The Send–a–solution function sends the Resolution freeform text from the problem record to the TSD knowledge database. Only solutions for problem records that contain a TSD record ID can be sent to TSD.

**Note:** A record with a TSD record ID means that either the record originated on TSD and was transferred to Information Management or else the record originated on Information Management, was transferred to TSD, and was returned to Information Management by some means (such as the Resume ownership function).

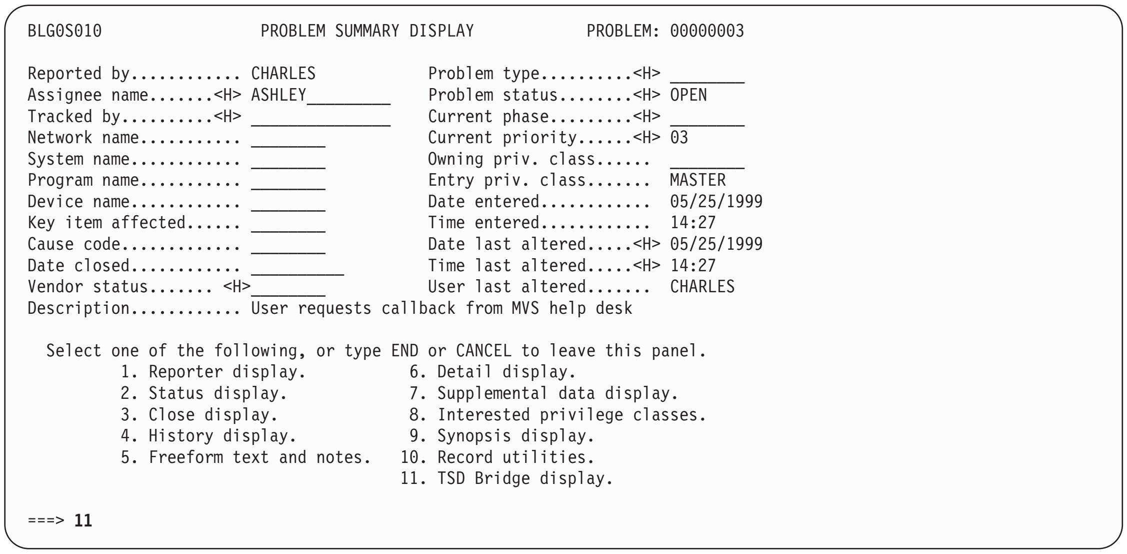
To send a solution, the Information Management user must display the record and select **3. Send a solution** from the TSD Bridge Display panel. If the problem record is owned by Information Management, contains a TSD record ID, and has a status of CLOSED, a request to send the solution for the problem record will be sent to TSD.

From a Tivoli Information Management for z/OS panel, type **display r 00000003** and press Enter to display the record.



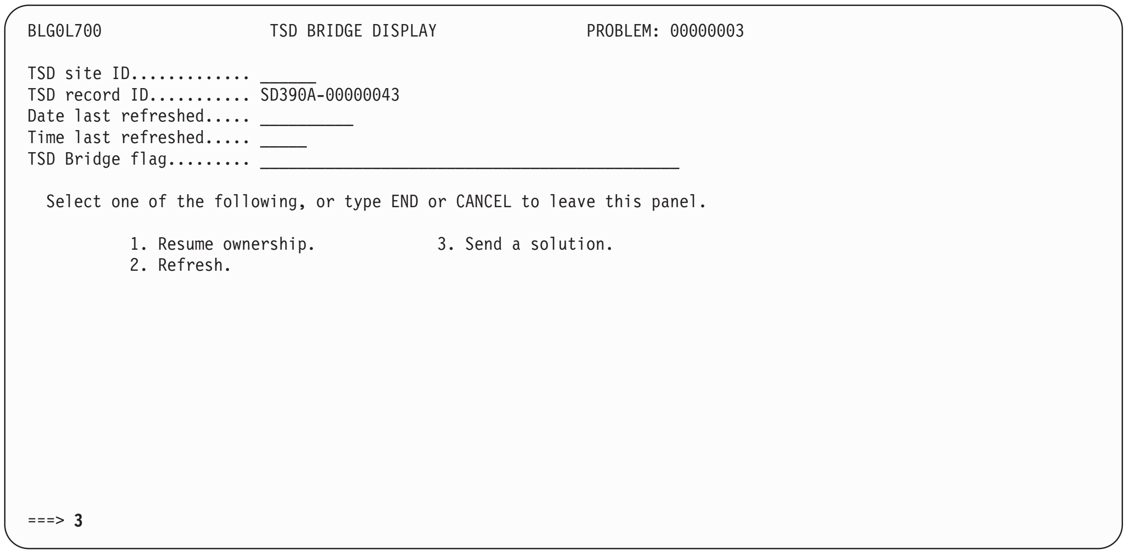
```
BLG0EN20              --- PRIMARY OPTIONS MENU ---     APPLICATION: MANAGEMENT

OPTIONS:

      1. OVERVIEW.......Display general information and product enhancements.
      2. PROFILE........Display or alter invocation or session defaults.
      3. APPLICATION....Change application, list available applications.
      4. CLASS..........Change current class, list available classes.
      5. ENTRY..........Create a record.
      6. INQUIRY........Search for records.
      7. UTILITY........Copy, display, print, delete, and update records.
      8. GLOSSARY.......Display a list of searchable words in the database.
      9. PMF............Modify or create panels.


           Select an option, enter a command, or type QUIT to exit.


          Tivoli Information Management for z/OS Version 7 Release 1
               5697-SD9 (C) Copyright IBM Corp., 1981, 2001.
===> display r 00000003
```

Type **11** and press Enter to select TSD Bridge display.



```
BLG0S010               PROBLEM SUMMARY DISPLAY           PROBLEM: 00000003

Reported by............ CHARLES           Problem type..........<H> ________
Assignee name.......<H> ASHLEY_________   Problem status........<H> OPEN
Tracked by..........<H> _______________   Current phase.........<H> ________
Network name........... ________          Current priority......<H> 03
System name............ ________          Owning priv. class......  ________
Program name........... ________          Entry priv. class.......  MASTER
Device name............ ________          Date entered............  05/25/1999
Key item affected...... ________          Time entered............  14:27
Cause code............. ________          Date last altered.....<H> 05/25/1999
Date closed............ __________        Time last altered.....<H> 14:27
Vendor status....... <H>________          User last altered.......  CHARLES
Description............ User requests callback from MVS help desk

  Select one of the following, or type END or CANCEL to leave this panel.
          1. Reporter display.          6. Detail display.
          2. Status display.            7. Supplemental data display.
          3. Close display.             8. Interested privilege classes.
          4. History display.           9. Synopsis display.
          5. Freeform text and notes.  10. Record utilities.
                                       11. TSD Bridge display.

===> 11
```

Type **3** and press Enter to send a request to TSD to transmit the solution for the problem record to the TSD knowledge database. The request is sent if the problem record is owned by Information Management, contains a TSD record ID, and has a status of CLOSED. While the request is being processed, the record is locked in the Information Management database, but the user can work with other records. When the request has completed, the record is unlocked.



```
BLG0L700                 TSD BRIDGE DISPLAY               PROBLEM: 00000003

TSD site ID............ ______
TSD record ID.......... SD390A-00000043
Date last refreshed..... __________
Time last refreshed..... _____
TSD Bridge flag......... _____________________________________________

  Select one of the following, or type END or CANCEL to leave this panel.

           1. Resume ownership.            3. Send a solution.
           2. Refresh.











===> 3
```

**Note:** The contents of the fields on this panel are explained on page 358.

## Tivoli Service Desk Bridge TSXs

The following TSXs are used by the Tivoli Service Desk Bridge.

**BLGTSDFP**

File people record (create, update, delete)

**Environment:**

Called from control panel BLM1B040 when filing a people record interactively or using the HLAPI Start User TSP transactions (HL14).

**Input:**

**Interactive:**

None

**API:**

- RNID_SYMBOL PDB

**Output:**

**Messages:**

None

**Return Codes (API only):**

*Table 42. BLGTSDFP Return Codes*

| HICARETC | HICAREAS | Description |
|---|---|---|
| 8 | 900 | Record ID parameter is required. |

**BLGTSDPO**

Resume problem record ownership

**Environment:**

Called from data-entry panel BLG0L700 when displaying a problem record interactively or using the HLAPI Start User TSP transaction (HL14). The

USER_NOTIFIED PDB in this TSX contains the Information Management user ID to be notified when the function has finished. Because the Assignee Name of the problem record will be updated with this user ID, a people record with a **Person role** of TSD390&TSD must exist for this user ID.

**Input:**

**Interactive:**
None

**API:**

- RNID_SYMBOL PDB
- USER_NOTIFIED PDB

**Output:**

**Messages:**
None

**Return Codes (API only):**

*Table 43. BLGTSDPO Return Codes*

| HICARETC | HICAREAS | Description |
|---|---|---|
| 8 | 900 | Record ID parameter is required. |
| 8 | 901 | User parameter is required. |

**BLGTSDPR**
Refresh problem record

**Environment:**
Called from data-entry panel BLG0L700 when displaying a problem record interactively or using the HLAPI Start User TSP transactions (HL14).

**Input:**

**Interactive:**
None

**API:**

- RNID_SYMBOL PDB

**Output:**

**Messages:**
None

**Return Codes (API only):**

*Table 44. BLGTSDPR Return Codes*

| HICARETC | HICAREAS | Description |
|---|---|---|
| 8 | 900 | Record ID parameter is required. |

**BLGTSDPS**
Send problem record solution

**Environment:**
Called from data entry panel BLG0L700 when displaying a problem record interactively or using the HLAPI Start User TSP transactions (HL14).

**Input:**

**Interactive:**
None

**API:**

- RNID_SYMBOL PDB

**Output:**

**Messages:**
BLG03148

**Return Codes (API only):**

*Table 45. BLGTSDPS Return Codes*

| HICARETC | HICAREAS | Description |
|---|---|---|
| 4 | 900 | Record is not in closed status. |
| 8 | 900 | Record ID parameter is required. |
| 12 | 900 | Record was not found. |

**BLGTSDPT**
Transfer problem record

**Environment:**
Called from control panel BLG1A111 when filing a problem record interactively or using the HLAPI Start User TSP transaction (HL14).

**Input:**

**Interactive:**
None

**API:**

- RNID_SYMBOL PDB

**Output:**

**Messages:**
None

**Return Codes (API only):**

*Table 46. BLGTSDPT Return Codes*

| HICARETC | HICAREAS | Description |
|---|---|---|
| 8 | 900 | Record ID parameter is required. |

**BLGTSDRQ**
Process TSD requests

**Environment:**
Called using the LINK control line from a TSX

**Input:**

**Function:**
Operation to perform. Valid values: FILE, REFRESH, RESUME, SENDSOLUTION, TRANSFER

**RNID**
Record identifier

**Record type:**
Type of record. Valid values: PEOPLE, PROBLEM

**Notify ID:**
User ID – userid to notify (optional)

**Output:**

**Messages:**

- BLG03083
- BLG03146
- BLG03147
- BLG03149

**Return Codes (API only):**

*Table 47. BLGTSDRQ Return Codes*

| HICARETC | HICAREAS | Description |
|---:|---:|---|
| 8 | 900 | Function code is not valid. |
| 8 | 901 | Record type is not valid. |
| 12 | 900 | SETTSDDATA control line error. |

**BLGTSDSM**
Send message

**Environment:**
Called using the HLAPI Start User TSP transaction (HL14).

**Input:**

**Messages:**
None

**API:**

- RNID_SYMBOL PDB
- MESSAGE_PANEL PDB
- USER_NOTIFIED PDB

**Output:**

**Messages:**
None

**Return Codes (API only):**

Table 48. BLGTSDSM Return Codes

| HICARETC | HICAREAS | Description |
|---:|---:|---|
| 8 | 900 | Message panel parameter is required. |
| 8 | 901 | User parameter is required. |
| 12 | 900 | Message panel was not found. |

# V — Integrating with Other Tivoli Products

29

# Integrating with Tivoli Inventory

This section describes how you can integrate your Tivoli Information Management for z/OS system with the Tivoli Inventory application to receive an extract of Tivoli Inventory data on your host database. The material provided in this section will help you to install and use the interface that is provided by Tivoli Information Management for z/OS to receive and use Tivoli Inventory data on your system. It is intended for users of both Tivoli Information Management for z/OS and Tivoli Inventory. For full details on how to set up and use the Tivoli Inventory application, refer to the *Tivoli Inventory User's Guide*. It is assumed that you have Tivoli Inventory installed and operational before you integrate Tivoli Information Management for z/OS with Tivoli Inventory.

## Overview of Tivoli Inventory

This section provides a high-level overview of the Tivoli Inventory scanning process and describes how the Tivoli Information Management for z/OS interface to Tivoli Inventory works.

Tivoli Inventory is a hardware and software inventory-gathering application designed to help system administrators and accounting personnel manage the complexity of PC and UNIX systems in a distributed client/server enterprise.

It enables users to:

- Maintain and upgrade hardware and software
- Monitor and record changes in software and hardware configurations
- Manage systems from a central point
- Access inventory information to perform system auditing functions

Tivoli Inventory is profile-based. That is, Tivoli Inventory scanning instructions are defined in *profiles*. In the context of the Tivoli Framework environment, profiles are linked to a Tivoli resource through a *profile manager*. The profile manager distributes the Tivoli Inventory profile to target machines or sites. The Tivoli Inventory profile passes scanning instructions to the scanner program that resides on each target machine. After Tivoli Inventory scans the target machines, it creates special files that contain information from the scan. Information such as the hardware, software, system configuration, and physical inventory is stored in files in a standard format called Desktop Management Task Force (DMTF) 2.0 Management Information Format (MIF).

Tivoli Inventory processes the information in the MIF files and sends it to the Tivoli Management Server, which acts as the ″hub″ for further activity. The server parses the data in the MIF files and sends the information to a relational database management system

(RDBMS) server, where it is stored in an open relational database management system called the Tivoli Inventory *configuration repository*. For the configuration repository you can use the RDBMS of your choice, such as Oracle, Sybase, or Microsoft® SQL Server.

Once Tivoli Inventory updates the configuration repository with the results of a scan, Tivoli Inventory users can access the information with the Tivoli Framework query feature. For example, users can query the configuration repository for all systems that have an outdated version of a software product that will need upgrading in the next year. The Tivoli Framework query feature consists of query libraries and queries. Tivoli Inventory queries are contained in query libraries which reside in policy regions on the Tivoli Framework desktop.

## Overview of the Interface to Tivoli Inventory

The purpose of the Interface to Tivoli Inventory is to enable a bulk transfer of data, based on Tivoli Inventory views, to Tivoli Information Management for z/OS. The data is extracted from Tivoli Inventory by using one of several supplied queries, and is captured to a file. The file, which contains data from the view, is read and parsed by the interface to Inventory (called the **i2i** program). If the data is recognized as a supported view, it will be mapped into Tivoli Information Management for z/OS s-words. The views supported are detailed on page 374. Attribute records for attributes in the supported views are supplied. The data is sent via the Tivoli Information Management for z/OS HLAPI client. The i2i program was developed on, and has been tested with, the Tivoli Information Management for z/OS HLAPI/NT client.

Using i2i, you can open an exported view of the data that resided in the Tivoli Inventory configuration repository. Once the data is sent to the Tivoli Information Management for z/OS database, you can query or display the data from a Tivoli Information Management for z/OS panel (BLG00030) or copy the data to other areas of Tivoli Information Management for z/OS, such as Problem Management or Change Management. You can, for example, use this data to perform an analysis of your overall computing environment to assist your service desk personnel. And because this information is already available, your service desk personnel will not need to enter all of the environmental data into Tivoli Information Management for z/OS.

Queries are supplied that you can install on the Tivoli Inventory Management Server in a Tivoli Information Management for z/OS query library. These queries are supplied to extract the Tivoli Inventory data from supported views of the data. The queries are run using the query facility of the Tivoli Management Server desktop to access information in the configuration repository. The query results can be exported as ASCII files to a specific Tivoli Information Management for z/OS HLAPI client workstation. When the query is run, the results must be exported with the required TAB delimiter to the Tivoli node where the Tivoli Inventory interface tool resides. This node must have the Tivoli Information Management for z/OS HLAPI client installed. The *Tivoli Inventory User's Guide* describes how to set up query libraries, run queries, and export query results.

From the Tivoli desktop, you can, through the use of Structured Query Language (SQL) functions, use the Tivoli Framework query facility to access information in the configuration repository. Query libraries reside in policy regions and contain queries. The queries specify which repository to search, which view or table within the repository to query, and what information to retrieve. Views can be created so that a group of information can be accessed with a single query.

The ASCII files containing query results are used as the input to the interface to Inventory program. The i2i tool can be used interactively or in batch mode. It is used interactively to:

- Edit the default mapping tables
- Open a file of extracted data to:
  - View the result of a mapped record prior to sending data to Tivoli Information Management for z/OS.
  - Send the view of data, whether complete or partial, to Tivoli Information Management for z/OS.

**Note:** Mapping of sample data should be tested after editing the mapping table.

Once the mapping tables are edited satisfactorily, preparation for batch mode includes:

- Creating a list of files for transfer
- Specifying the batch log file name

At this point, a scheduler should be used to automate:

- Running Tivoli Inventory scans
- Running Tivoli Inventory queries
- Running i2iBatch

Once the data is in the database, you can display or query the data by selecting the SERVICDESK application from the Application Selection menu on panel BLG00030 on the Tivoli Information Management for z/OS host. You can also copy the data for use by other Tivoli Information Management for z/OS applications (such as Problem Management or Change Management) by using user exits BLG01273 and BLG01439. You can use these user exits to copy only the data you want to your customized application panels.

Special data model records—data view and data attribute records—are shipped with Tivoli Information Management for z/OS to define the supported Tivoli Inventory views and associated data. These records are provided on the base product installation media, and must be installed on a Tivoli Information Management for z/OS host database before you can receive and browse the Tivoli Inventory data. The i2i program maps the data extracted from Inventory to Tivoli Information Management for z/OS fields and loads the data into the database through the client API using these data view records. Database 6 is reserved for the purpose of storing data extracted from Tivoli Inventory.

## Components of the Interface to Tivoli Inventory

### Host Components

The following host components are provided with Tivoli Information Management for z/OS to support the interface to Tivoli Inventory:

- Data model records (data view and data attribute records)
- JCL to load the data model records into the SDDS
- Dictionary entries (s-words and p-words)
- Panels to display and query the inventory data in the Tivoli Inventory database
- Panels to delete the inventory records from the Tivoli Inventory database

- Predefined queries to search the inventory data in the Tivoli Information Management for z/OS database

  **Note:** These are queries that you can use from the Tivoli Information Management for z/OS panels, which display inventory data from the Tivoli Information Management for z/OS database. (These queries should not be confused with the Tivoli Inventory queries that are shipped with Tivoli Information Management for z/OS and installed on the Tivoli Management Server to support data extraction for Tivoli Information Management for z/OS.)

- Report format tables (RFTs) to print inventory records in reports

## Workstation Components

The following workstation components are provided with Tivoli Information Management for z/OS to support the interface to Tivoli Inventory.

| | |
|---|---|
| ■ i2i_queries.sh | The i2i_queries.sh is a Bourne shell script which creates a query library containing the Service Desk queries. It must be run by an administrator with proper Tivoli authorization in the appropriately configured Tivoli Managed Resource (TMR) where the library resides. Queries are created with the same name as the supported Inventory views.<br>**Note:** Make sure that the preceding queries do not already exist; if they do exist, the new query will not be created. |
| ■ Blmi2i.jar<br>■ Blmtsd12.jar<br>■ Blminfoapi.jar<br>■ Extend.jar | Contains the Java™ classes for the i2i program. |
| ■ i2i.bat | Batch command file to start the interactive version of the i2i program. The interactive version allows you to modify map files, open scan files and view and/or map an individual record and load the records to the Tivoli Information Management for z/OS database. |
| ■ i2iBatch.bat | Batch command file to start the batch version of the i2i program. The batch version accepts the name of scanfile output from one of the supplied queries. The data file is loaded into the Tivoli Information Management for z/OS database |

## Installing the Interface to Inventory

This section lists the prerequisites for installing the interface to Inventory and describes how to install the necessary components to support the interface. Tivoli Information Management for z/OS relies on Tivoli Inventory and the Tivoli Framework. Before you install the host and workstation components to support the interface, ensure that Tivoli Inventory is installed and that target machines can be scanned by Tivoli Inventory. For information on installing Tivoli Inventory, refer to the *Tivoli Inventory User's Guide*.

Installation involves loading both host and workstation components. Although it is not necessary to perform the host installation tasks before installing the workstation components, it is recommended.

## Installing the Host Components

Follow these steps to install the host portion of the interface to Inventory in Tivoli Information Management for z/OS. The steps involve loading the data model records supplied on tape, creating the Tivoli Inventory database, and updating your session-parameters members to access the newly created database. It is assumed that you have already installed Tivoli Information Management for z/OS Version 7.1.

1. Load the Tivoli Information Management for z/OS data model records (data view and data attribute records) from the SBLMRCDS data set library. Use batch job BLHRCDSJ in the SBLMSAMP sample library to load these records. The *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* contains additional information about loading data model records.

   Before you can run the batch job, you must edit the JCL. Instructions are provided in the JCL comments. In your input data stream, be sure to specify BLHLRINV as the name of the PDS member containing the list of data model records to be loaded.

   The BLHRCDSJ job unflattens the data model records and adds them to the Tivoli Information Management for z/OS database you specified.

   You should also load the records listed in the BLHLRBAS list if they have not already been loaded.

2. Create the SDDS and the SDIDS for the Tivoli Inventory database. The SDDS will be the Tivoli Information Management for z/OS database that will hold the data records extracted from Tivoli Inventory. The SDIDS for the Tivoli Inventory database must be defined with a key length of 34. Use the AMS DEFINE CLUSTER command to create the SDDS and the SDIDS. Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for instructions on defining the SDDS and SDIDS.

3. Create the session-parameters member that will be used to load the Tivoli Inventory database with the data extract. At a minimum, you will need to use the BLGPARMS macro to define the session parameters, and the BLGCLUST macro to define the database. The *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* provides details on how to use these macros.

   a. In the BLGPARMS MODELDB keyword, specify the name of the database that contains the data model records supplied with Tivoli Information Management for z/OS.

   b. Dates are sent from Tivoli Inventory through the Tivoli Information Management for z/OS HLAPI client in the external date format of MM/DD/YYYY. If this is not the primary external date format that you use in Tivoli Information Management for z/OS, you must perform one of the following steps:

      - If your primary external date format has a length of 10 characters, use the i2i program to change the mapping of the Tivoli Inventory date to your external date format.

      ***or***

      - If your primary external date format is not 10 characters long, use the BLGPARMS DATECNV keyword and specify your primary external date format, specify MM/DD/YYYY as the secondary date format, and specify PRIMARY. For example:

        ```
        DATECNV=(BLGCDATS,YYYY/MM/DD,YY/MM/DD,MM/DD/YYYY,PRIMARY)
        ```

In this example, YYYY/MM/DD is the internal date format. Dates can be entered in MM/DD/YYYY format but stored in the primary external date format of YY/MM/DD.

c. Use the BLGCLUST macro to add the Tivoli Inventory database. You must specify NAME=5 as the external name of the Inventory read/write database.

4. Update your session-parameters members so they can access the Tivoli Inventory database you just created:

   a. Use the BLGPARMS macro and specify, in the MODELDB keyword, the name of the database that contains the data model records supplied with Tivoli Information Management for z/OS for use with the interface to Inventory. (Use the name that was set up in Step 3a on page 373.)

   b. Use the BLGCLUST macro to add the Tivoli Inventory database. You must specify NAME=6 as the external name of the Tivoli Inventory read-only database.

When you have completed these steps, you can proceed with installing the workstation portion of the interface.

## Installing the Workstation Components

You should ensure that the following prerequisites have been met before proceeding with installation:

- Tivoli Inventory Version 3.6
- HLAPI for Windows NT

The installation CD-ROM contains a directory **/Tivoli_Int/INV**. Run SETUP.EXE from within this directory to install i2I on a workstation.

Upgrades or patches that can be downloaded from a Tivoli Web site may be available for HLAPI for Windows NT or for the Interface to Tivoli Inventory. Visit the Web site at `http://www.tivoli.com/TSD390` for more information.

### Install Queries in Query Library

On the Tivoli Management Server, run the following shell script file to create the Tivoli Inventory queries needed by Tivoli Information Management for z/OS to extract data from the configuration repository. This file was created as a result of running SETUP.EXE from within the **/Tivoli_Int/INV** directory.

```
\i2i\setup\i2i_queries.sh
```

When you run this script file, a new query library called **ServiceDeskQueries** is created on the Tivoli Management Server. The following queries are automatically added to the **ServiceDeskQueries** library:

**Hardware Views**

- PC_MEMORY_VIEW
- INVENTORYDATA
- PC_PORTS_VIEW
- LOGICALDRIVE_VIEW
- INSTALLED_COPROCESSOR_VIEW
- INSTALLED_HARDDISK_VIEW

**Software Views**

- PC_BIOS_VIEW
- NT_INFO_VIEW
- INSTALLED_SOFTWARE_VIEW
- INSTALLED_CONFIG_FILE_VIEW

**Network Views**
- PC_LAN_CONN_VIEW
- NETWARE_SERVER_VIEW
- NETWORK_NODE_VIEW
- PC_LAN_VIEW

These queries also correspond to the queries that you see when using the i2i program on the Tivoli Information Management for z/OS HLAPI client workstation.

## Installation Considerations

These are some additional items that you should consider:

- Make sure that you set up your MRES to use pre–started MRES sessions (refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide*). In the MRES parameters for the pre–started MRES, you must specify `BYPASS_PANEL_PROCESSING=YES`. If this is not set up properly, you may receive a message indicating that your session member is not valid.
- Test the HLAPI client with the supplied Java test program.
- Make sure that you set Preferences for Initialization and the Batch Files.
- If you modify the supplied Inventory queries, the query name must match one of the supported Inventory views. The ASCII file output from the query contains the query name. This name is used to verify the file in order to use the appropriate mapping table.

## Scan Target Machines

Now, you are ready to scan target machines using Tivoli Inventory. You can perform this scan from any workstation with a Tivoli Inventory desktop. Complete the scan according to the instructions provided in the *Tivoli Inventory User's Guide*.

## Verify Installation

To verify that the workstation installation was performed correctly, check the query libraries from a Tivoli desktop. An icon for the **ServiceDeskQueries** library that was created is displayed.

1. Use the Tivoli Framework query facility to run the INVENTORYDATA query in the **ServiceDeskQueries** library. (Refer to the *Tivoli Inventory User's Guide* for instructions.)

   The results of the query will display in a dialog. If no data appears, either the workstation portion of the installation did not complete successfully, or else User Data Forms were not submitted prior to the scan. Review the steps you followed and repeat the workstation installation process as necessary.
2. Save the results of the query either on the local machine or by exporting the query results to the Tivoli Information Management for z/OS client workstation.

   **Note:** To export the results of a Tivoli Inventory query to your HLAPI client workstation, your client workstation must have the Tivoli Management Agent installed to allow it to be managed by Tivoli Management software. The Tivoli Information Management for z/OS HLAPI client does not provide the agent software.

3. Now, from the Tivoli Information Management for z/OS client workstation, follow these steps:
   a. Run the i2i program (**i2i.bat**) installed on the workstation.

      The Tivoli Information Management for z/OS interface to Inventory window displays.
   b. Select **File->Open** to browse the ASCII file you saved (or exported to your workstation) from Tivoli Inventory. The view corresponding to the ASCII data is highlighted (INVENTORYDATA), and the file is displayed along with its Tivoli Information Management for z/OS mapping table. In order to parse properly, the file requires two things:
      - Line 1 contains **Query Name :** *viewname* (where *viewname* is the supported view); for example,
        ```
        Query Name : INVENTORYDATA
        ```
      - Line 2 and all subsequent lines are separated with the tab delimited, where line 2 contains the column titles and lines 3 and beyond contain the data.

      If this does not occur, an error message panel informs you of any error found. Review the error and take the appropriate corrective action.
   c. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* and take the actions described to use pre-started MRES sessions. Ensure that you have specified `BYPASS_PANEL_PROCESSING=YES` in the MRES parameters for the pre-started MRES.

## Customizing the Interface to Inventory

During the installation of the host and workstation portions of the interface to Inventory, you can customize the following:

- On the host, if you want to copy the inventory data that is loaded in the Tivoli Information Management for z/OS database to other applications (such as to your Problem Management or Change Management application), you must modify your panels to use program exit BLG01273 (Add Data from One Record to Another Record), or program exit BLG01439 (Extended Data Copy).

  Program exit BLG01273 obtains information from one record and automatically adds it to the record being created or updated. Program exit BLG01439 performs all the functions of exit BLG01273, but can also validate data against an assisted-entry panel and copy single items into list processor data. For information on how to use these program exits, refer to the *Tivoli Information Management for z/OS Panel Modification Facility Guide*.

  You should not attempt to modify the data model records that are shipped with Tivoli Information Management for z/OS to support the interface to Inventory.
- On the Tivoli Information Management for z/OS client workstation, you can use the i2i program to edit the mapping of Inventory data to Tivoli Information Management for z/OS. When you edit the mapping tables, you have default maps which map into s-words. You can edit the maps; you cannot edit the s-words.

## Using the Interface to Inventory

In order to use Inventory records, you must have Config Record Display authority. In order to delete Inventory records, you must have DBADMIN (Database Administration) authority. Authority levels are described in the *Tivoli Information Management for z/OS Program Administration Guide and Reference*.

### Using the Interface from a Workstation

- Select **PREFERENCES -> INITIALIZATION** to edit the SECURITY_ID, APPLICATION_ID, PASSWORD, and DATABASE_PROFILE. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for specific information on these control PDBs. To edit an item, highlight the item and select **Edit**. Supply the new value and select **OK** (or **CANCEL**). The updates are saved in the i2i subdirectory of the user's home directory. After returning to the Preferences panel, select **SAVE** (or **CANCEL**).
- If you want to edit a mapping table, click on the **INVENTORY VIEW TREE** and select the view you wish to edit. A single left-mouse click will display the map. To edit the map for a specific s-word, highlight the row and select **EDIT**. Remember to save the edited map. The updated maps are saved in the i2i subdirectory of the user's home directory.
- If you would like to verify the updated map, select **FILE->OPEN** and open a previously created ASCII scan file for the specific view. Verify that the appropriate map is loaded and that the file is parsed properly. Select a row and click on **VIEW** to see the fields as represented in the Inventory database. Select **MAP** to see the fields as they will be mapped to s-words in Tivoli Information Management for z/OS.

  **Note:** The Client log is a cumulative log, so you should discard old messages prior to transmitting new views.
- If you would like to "send" a view to Tivoli Information Management for z/OS, select **FILE->OPEN** and open a previously created ASCII scan file for a view. Again, verify that the appropriate map is loaded and that the file is parsed properly. Make sure that the Tivoli Information Management for z/OS Client Requester is running. Select **FILE -> SELECT ALL** to send the entire View.

  **Note:** If you send less than a complete view, the result may be databases that are not synchronized. The status window will be updated during transmission. The terminated message will display transaction error counts. The history icon may be selected for a complete status report. You can also refer to the client log (specified in your client database profile) for detailed messages.

  Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for additional information. Select **SEND**.
- The first item in the history log specifies whether the HL01 was successful. The next set of rows specifies the result of each transaction. The last item specifies the status of the HL02. Please refer to the Client log, the name of which is specified in the client database profile, for detailed Tivoli Information Management for z/OS transaction messages.
- If you plan to use batch, select **PREFERENCES->BATCH->BATCH INPUT AND BATCH OUTPUT**. Update the list of input files by selecting **ADD**, **EDIT**, or **DELETE**. The files should exist (although not necessarily at the time of the creation of the list) and contain proper scan data at the time that the i2iBatch.bat is run. The output

files specify the name of the old and new logs. The batch log will contain the date and time that the job was run, the scan file name, the view name, and the status messages. For detailed messages, refer to the Tivoli Information Management for z/OS client log. The logs are written to the i2i subdirectory of the user's home directory.

## Using the Editor

The edit panel displays two panels:

- The tree on the left represents the parsed mapping table for the selected view. The s-words are the first branches of the tree and the lower branches represent nested functions. The mapping table ″maps″ data to the specified s-word. The data field may contain new data, a Tivoli Inventory column title, or a nested function.
- The panel on the right is the editor panel. This panel enables editing the map for a specific s-word. If the s-word was selected from the tree on the left or if the s-word is highlighted prior to going to the mapping editor, the map may be edited ″free form″. In other words, you may edit the map without editor panel assistance. If you edit free form, substitute the following variables for the special characters: `I2iLPAREN` for the left parenthesis "(" character or `I2iRPAREN` for the right parenthesis character " )" or `I2iCOMMA` for the comma "," character. Spaces are not allowed between the function arguments. **Click on child nodes, if any, to edit function calls** will be selected if an s-word was selected from the tree. If a nested function was selected, a specific function editor panel will be displayed containing the current function arguments. The selected nested function will be highlighted in the map field. The function arguments may be modified. Assistance is provided by the use of spin buttons or radio buttons. If the data argument is expected to be the result of a nested function, the button **The data argument will be determined by the evaluation of the nested function** will be selected. If the data argument is specified, the radio button **The data argument is provided** will be selected. If several arguments are listed, a list box will be provided. Update the box by selecting the **ADD**, **EDIT**, or **DELETE** buttons. When the edit of the map is complete, select **Apply** or **Reset**. If **Default** is selected, the original default map for that specific s-word will be retrieved.

  **Note:** Using the editor panels is the preferred method of editing. This method provides verification, whenever possible, and will automatically substitute the special variables. If you choose to edit free form, once the node has been removed, recalculated, and replaced, it is a good idea to click on the tree nodes representing any nested functions to recapture the verifications.
- The Editor Help menu lists the current supported functions. The function call, description, and examples are provided. The **OK** button provides **FILE->SAVE** and **FILE->CLOSE** functions. The function tabs may be selected for viewing.

When the selected view mapping table edit session is complete, select **FILE->SAVE** or **FILE->CLOSE** (**Close** will prompt for **Save**). If saved, the updated map will be saved in the i2i directory of the user's home directory.

- To retrieve the entire default map for the selected view: select **FILE->DEFAULT MAP**
- To print the map being edited: select **FILE->PRINT->PRINT MAP**
- To print the original default map for the selected view: select **FILE->PRINT->PRINT DEFAULT MAP**

Printing is enabled to devices supported by the default graphics object.

## Editor Functions

These are the currently available functions of the editor. If the functions receive arguments in a format other than specified, the data may be returned unchanged.

**change(data,source1,target1,source2,target2,...)**

The string given by source1, where found within data, is changed to the string given by target1. It then changes the string given by source2, where found in the result of the first operation, to the string given by target2. This continues until all source strings have been processed. If the last matching target string is missing, it defaults to a null string. For example, if fieldName is abcdefghijklm then

```
change(fieldName,abc,def,def,ghi)
returns
ghighighijklm

change(abcdefghijklm,def)
returns
abcghijklm
```

**fromIMDate(IMDate)**

A date in the format mm/dd/yy is converted to mm/dd/yyyy, where year characters 50 through 99 represent the years 1950 through 1999 and year characters 00 through 49 represent the years 2000 through 2049. For example:

```
fromIMDate(12/05/49)
returns
12/05/2049
```

**fromIMPriority(IMPriority)**

Values 6 through 99 are mapped to 5, while not altering values 0 through 5. For example:

```
fromIMPriority(21)
returns
5

fromIMPriority(0)
returns
0
```

**fromIMTime(IMTime)**

The military time format of hh:mm is converted to a time in the format hh:mm:ss am or hh:mm:ss pm by adding a seconds field of 00 and am or pm. For example:

```
fromIMTime(12:34)
returns
12:34:00 pm
```

**nullDefault(data,defaultValue)**

The value defaultValue is returned when data is a null string. Otherwise, the value for data is returned. For example, if fieldName is abc, then:

```
nullDefault(fieldName,default)
returns
abc

nullDefault(,fieldName)
 returns
abc

nullDefault(,)
returns
""
```

**stripLeading(data,stripCharacters)**

This function strips the leading characters from the target data. For example, if fieldName is 0000200, then:

```
stripLeading(fieldName,0)
returns
200

stripLeading(fieldName)
returns
0000200

stripLeading(000000,0)
returns ""

stripLeading(wordwordzz,word)
returns
zz
```

**subString**

This function returns a specified substring of the target data. The first character is position 1. The formats are:

subString(data,startPosition)
subString(data,startPosition,length)
subString(data,startPosition,length,padCharacter)

**startPosition**
is the Starting index position of the substring. If the index is beyond the end of the string, the function returns a null string.

**length** is the length of the substring. If the substring extends beyond the end of the string, the substring is padded with the character given by the padCharacter argument. If length is not specified, the substring goes from the starting position to the end of the string.

**padCharacter**
is the character to use as padding if the substring extends beyond the end of the string. The default pad character is a single space.

For example, if fieldName is abcdef then:

```
subString(fieldName,2,3)
returns
bcd

subString(fieldName,4,5)
returns
def + 2 spaces

subString(subString(abcdef,2,3),2,1)
returns c

subString(abcdef,7,1)
returns
""
```

**toIMDate(foreignDate)**

A date in the format mm/dd/yyyy is converted to mm/dd/yy, where year characters 50 through 99 represent the years 1950 through 1999 and year characters 00 through 49 represent the years 2000 through 2049. For example:

```
toIMDate(01/31/1950)
returns
01/31/50
```

**toIMTime(foreignTime)**

A time in the format hh:mm:ss am or hh:mm:ss pm is changed to military time hh:mm. For example:

```
toIMTime(12:34:56 am)
returns
00:34
```

**translate(data,inputCharacters,outputCharacters)**

The characters given by **inputCharacters**, where found within data, are changed to the characters given by **outputCharacters**. If not specified, **outputCharacters** defaults to spaces. For example, if **fieldName** is a,b,c,d then:

```
translate(fieldName,I2iCOMMA, )
returns
a b c d

translate(aI2iCOMMAbI2iCOMMAcI2iCOMMAd,fieldName)
returns
7 spaces
```

**translateWord(data,sourceWord1,targetWord1,sourceWord2,targetWord2,...)**

The string given by **sourceWordN** is changed to the string given by **targetWordN**, if found within data. If the last matching target is missing, it defaults to a null string. If **sourceWord1** is *, any result for data is a match. For example, if **fieldName** is word then:

```
translateWord(fieldName, fieldName, bird)
returns
bird

translateWord(word,word)
returns
""
```

**words(data,firstWord) or words(data,firstWord,numberOfWords)**

Words are returned from data beginning with **firstWord**, where words are separated by spaces. If the index given by **firstWord** is not valid, a null string is returned. The **numberOfWords** argument can be used to specify how many words to return. If the **numberOf words** argument is not specified, all words to the end of the string are returned. For example, if **fieldName** is a b c d then:

```
words(fieldName,2,1)
returns
b

words(subString(fieldName,3,3),2,2)
returns
c

words(a b c d,5,1)
returns
""
```

## Using the Interface from the Host

This sections depicts the various hardware and software data that can be viewed from the host. The inventory records are stored in database 6 with system-assigned RNIDs.

**Note:** Before using the search panels provided by this interface, make sure that you have the *Quick search?* field in your User Profile set to **YES**. The *Tivoli Information Management for z/OS User’s Guide* contains information about setting values for your User Profile.

To display the Tivoli Information Management for z/OS Tivoli Inventory menu, type **3** and press Enter from the primary options menu BLG0EN20.

```
BLG0EN20              --- PRIMARY OPTIONS MENU ---     APPLICATION: MANAGEMENT

OPTIONS:

      1. OVERVIEW.......Display general information and product enhancements.
      2. PROFILE........Display or alter invocation or session defaults.
      3. APPLICATION....Change application, list available applications.
      4. CLASS..........Change current class, list available classes.
      5. ENTRY..........Create a record.
      6. INQUIRY........Search for records.
      7. UTILITY........Copy, display, print, delete, and update records.
      8. GLOSSARY.......Display a list of searchable words in the database.
      9. PMF............Modify or create panels.


           Select an option, enter a command, or type QUIT to exit.


          Tivoli Information Management for z/OS Version 7 Release 1
               5697-SD9 (C) Copyright IBM Corp., 1981, 2001.
===> 3
```

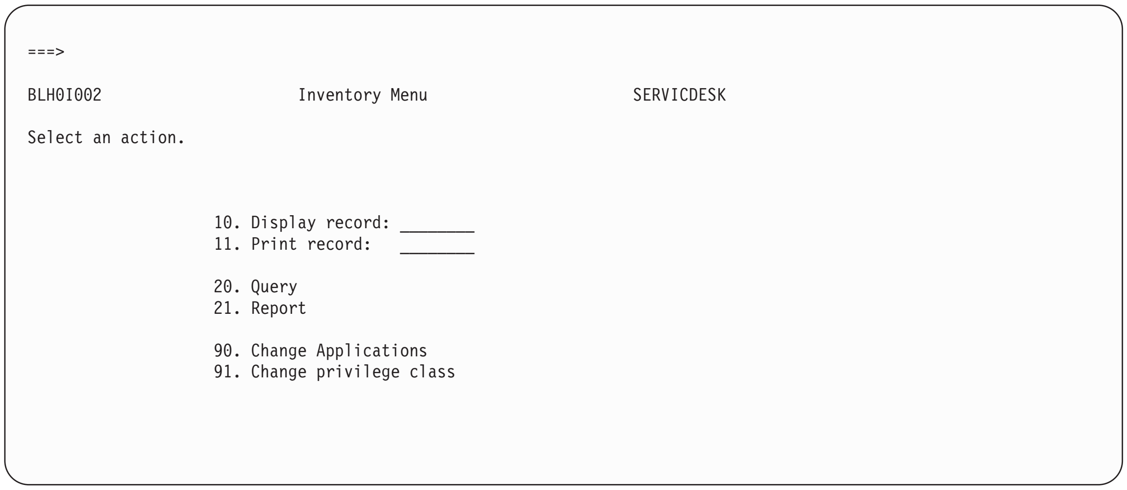
Type **9** and press Enter to display the Inventory Menu.

```
===>

     + BLG00030 ------------ APPLICATION SELECTION ------------- 1 OF 1-+
     |                                                                  |
     |             IDENTIFY THE APPLICATION FOR THIS SESSION            |
     |                                                                  |
     |                                                                  |
     | OPTIONS:                                                         |
     |                                                                  |
     |       1. SYSTEM.......Use System dialogs for entry/inquiry.      |
     |       2. MANAGEMENT...Use Management dialogs for entry/inquiry.  |
     |                                                                  |
     |       4. INTEGRAT.....Use Integration Facility dialogs.          |
     |                                                                  |
     |       9. SERVICEDESK...Use Consolidated Service Desk.            |
     |                                                                  |
     +---------------------- SELECT APPLICATION ------------------------+



===> 9
```

At the Inventory Menu, BLH0I002, type **20** and press Enter to begin the query process; depending on your authority level, this panel may display an additional choice:

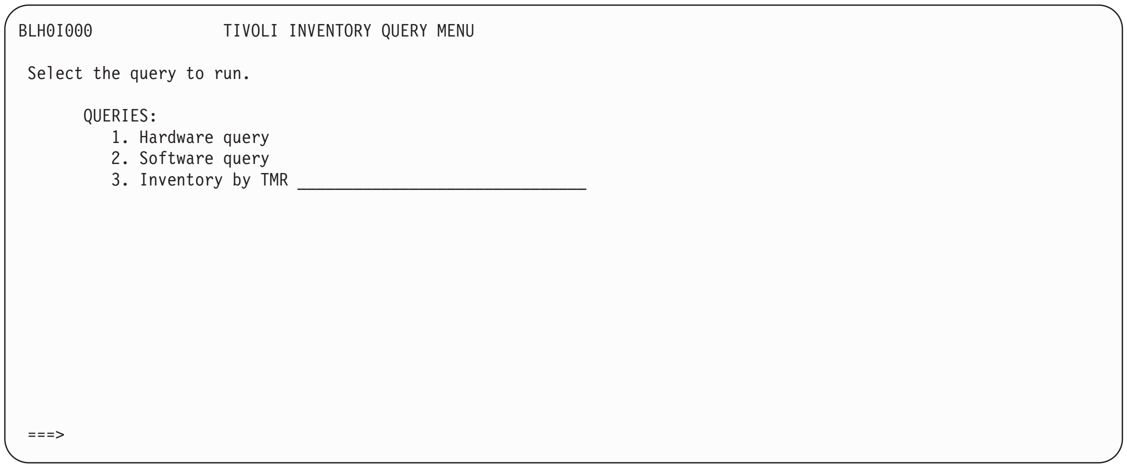
```
12. Delete record: ______
```

To return to panel BLG00030, type **90** and press Enter on panel BLH0I002.



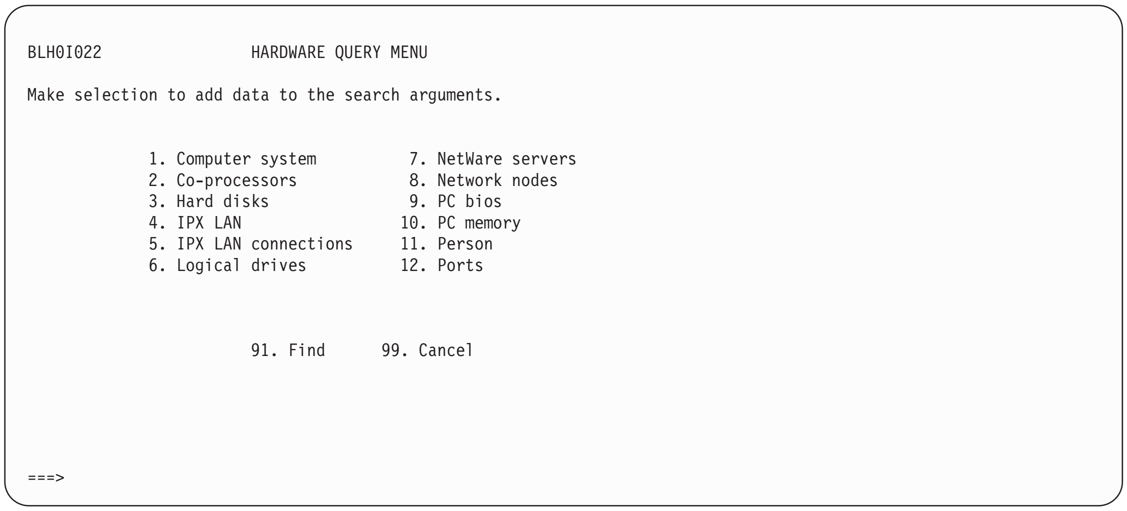
```
===>

BLH0I002                   Inventory Menu                    SERVICDESK

Select an action.



                    10. Display record: ________
                    11. Print record:   ________

                    20. Query
                    21. Report

                    90. Change Applications
                    91. Change privilege class
```

From the Tivoli Inventory Query Menu, BLH0I000, you can choose to initiate a hardware query, a software query, or query Inventory by a TMR. If you select **1** from this screen, panel BLH0I022 on page 383 shows the hardware queries available; if you select **2** from this screen, panel BLG0I023 on page 384 shows the software queries available.



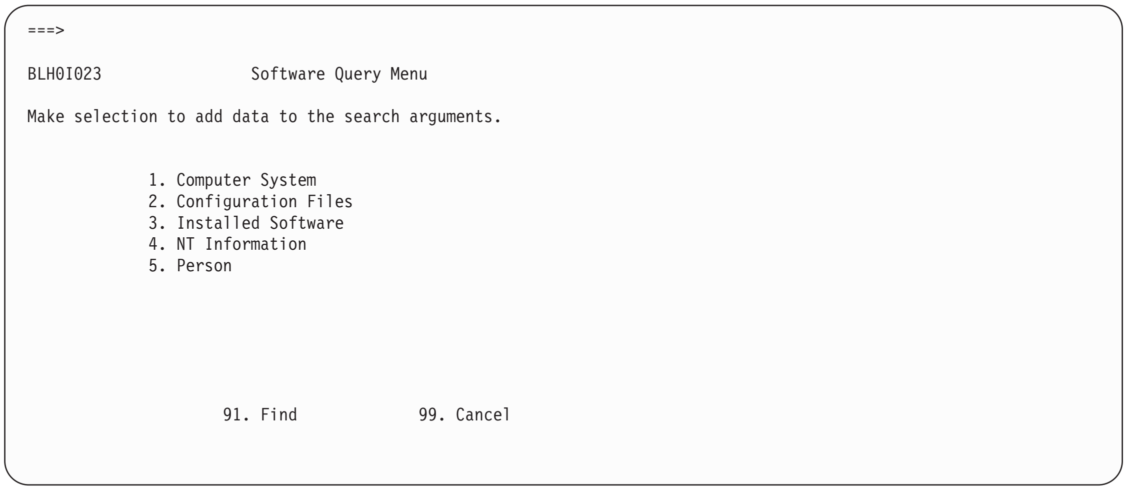
```
BLH0I000             TIVOLI INVENTORY QUERY MENU

 Select the query to run.

       QUERIES:
          1. Hardware query
          2. Software query
          3. Inventory by TMR _______________________________















 ===>
```

From panel BLH0I022, you can select an item, and on subsequent panels, add search arguments in order to obtain information about the hardware that is installed.



```
BLH0I022                HARDWARE QUERY MENU

Make selection to add data to the search arguments.


             1. Computer system          7. NetWare servers
             2. Co-processors            8. Network nodes
             3. Hard disks               9. PC bios
             4. IPX LAN                 10. PC memory
             5. IPX LAN connections     11. Person
             6. Logical drives          12. Ports



                        91. Find      99. Cancel





===>
```

From panel BLH0I023, you can select an item and on subsequent panels, add search arguments in order to obtain information about the software that is installed.



```
===>

BLH0I023               Software Query Menu

Make selection to add data to the search arguments.


             1. Computer System
             2. Configuration Files
             3. Installed Software
             4. NT Information
             5. Person






                     91. Find             99. Cancel
```

## Messages

You may receive error messages and warning messages from Tivoli Inventory; the content of these message will explain cause and corrective action. In addition, these are the status messages that can be issued from Tivoli Inventory:

### Status Messages

These are the transaction status messages:

**Initialization: BAD.**
Possible causes:

- Tivoli Information Management for z/OS Requester not running.
- Pre-started MRES sessions not set up properly.
- MRES not running.
- Tivoli Information Management for z/OS not set up properly or not running.
- SECURITY_ID, APPLICATION_ID, or PASSWORD is not valid.
- DATABASE_PROFILE or profile contents are not valid:
  - Host name (ping host name, check address)
  - MRES (check TCP/IP SERVICES for server name and port)

**Initialization: GOOD.**
The HL01 transaction completed successfully.

**Initialization Error:**
An exception was thrown during the initialization step. The exception message is displayed.

**ID QueryError:**
Where ID is a specific HARDWARE_SYSTEM_ID. A query transaction was performed to determine if the record should be created or updated, but an exception occurred. The exception message is displayed.

**ID RNID Record created.**
Where ID is a specific HARDWARE_SYSTEM_ID and RNID is the Tivoli Information Management for z/OS RNID. The record id did not previously exist and the transaction was successful.

**ID CreateException:**
Where ID is a specific HARDWARE_SYSTEM_ID. An exception was thrown during a create transaction. The exception message is displayed.

**ID RNID Record not created.**
Where ID is a specific HARDWARE_SYSTEM_ID and RNID is the Tivoli Information Management for z/OS RNID. The record id did not previously exist and the transaction was not successful. Refer to the client log for the Tivoli Information Management for z/OS return and reason codes.

**ID RNID Record updated.**
Where ID is a specific HARDWARE_SYSTEM_ID and RNID is the Tivoli Information Management for z/OS RNID. The record id did previously exist and the transaction was successful.

**ID RNID Record not updated.**
Where ID is a specific HARDWARE_SYSTEM_ID and RNID is the Tivoli Information Management for z/OS RNID. The record id did previously exist and the transaction was not successful. Refer to the client log for the Tivoli Information Management for z/OS return and reason codes.

**Terminating Session.**
The session is terminating.

**Terminated...Transaction Errors:**
The session has terminated with the specified error count.

30

# Integrating with Tivoli Enterprise Console (TEC)

One of the highest priorities of information technology (IT) departments charged with managing distributed computing environments is to ensure that problems, as well as conditions that could lead to problems, are handled in a timely and efficient manner. Ensuring the high availability of applications running in these environments is another essential responsibility of the IT staff to its users. Without the proper management platform and tools for generating, transmitting, processing, and responding to significant problem alerts, it is impossible to deploy reliable advanced client/server technologies for critical business applications. The Tivoli Enterprise Console (TEC) provides the tools for handling problems within a distributed computing environment.

Tivoli Information Management for z/OS provides the TEC Integration Facility which integrates the Tivoli Enterprise Console into the Tivoli Information Management for z/OS problem management application. The Tivoli Information Management for z/OS TEC Integration Facility is described in the following sections.

## TEC Integration Facility

The TEC Integration Facility enables you to create, update, and delete Tivoli Information Management for z/OS problem records based on events received by the Tivoli Enterprise Console (TEC). Specifically, you can:

- Issue problems to Tivoli Information Management for z/OS from the TEC based on rules that apply to the TEC events.
- Issue problems to Tivoli Information Management for z/OS from the TEC manually
- Modify events in the TEC and automatically update the corresponding problem records in Tivoli Information Management for z/OS.

## Installing the TEC Integration Facility

The TEC Integration Facility uses Problem Service to pass information from TEC events to Tivoli Information Management for z/OS problem records. Problem Service must be installed on a Tivoli Management Region (TMR) server. A TEC server must also be running in the same TMR and the TEC Integration Facility must be installed on the same workstation as the TEC server. The TEC Integration Facility requires the following software:

- Tivoli Enterprise Console Version 3.1
- Problem Service (described in “III — Problem Service” on page 247)
- Patch 3.0-TMP-0012 for the Tivoli desktop may be required if you cannot display the tasks in a task library from within the Enterprise Console.

The TEC Integration Facility is supported on the following platforms:

- AIX 4.2
- Windows NT 4.0

The TEC Integration Facility is shipped on the Tivoli Information Management for z/OS installation medium in the **\Tivoli_Int\TIF** directory. If you are installing this from a CD-ROM, insert the CD-ROM into your CD-ROM drive. Then use the Tivoli desktop to install the TEC Integration Facility by clicking **Install Product**. The files used by the TEC Integration Facility are copied to the **$BINDIR\INFOTEC** directory. **$BINDIR** is the directory in which Tivoli is installed. See “List of Files” on page 394 for a list of the files that are copied.

## Task Library

After installing the TEC Integration Facility a new task library called INFOTEC is added in the TMR’s policy region. This task library contains three tasks: Create_Record, Update_Record, and Delete_Record. You can modify these tasks to meet your requirements, or you can add your own tasks to the task library.

The Create_Record task extracts information from a selected TEC event and then creates a problem record in the Tivoli Information Management for z/OS database. The task uses the event number from the event to relate the event to a problem record. See “Mapping Event Data to Problem Records” on page 391 for more information about this relationship.

The Update_Record task extracts information from a selected TEC event and then updates the appropriate problem record in the Tivoli Information Management for z/OS database. The record to be updated must have been created previously using either the Create_Record task or the Trouble Ticket task. See “Trouble Ticket” on page 389 for more information about the Trouble Ticket task.

The Delete_Record task extracts information from a selected TEC event and then deletes the appropriate problem record in the Tivoli Information Management for z/OS database. The record to be deleted must have been created previously using either the Create_Record task or the Trouble Ticket task.

To test the tasks in the task library, generate an event by entering the TEC **wpostemsg** command from the command prompt. The format of this command is:

**wpostemsg -r** *rrr* **-m** *mmm slot1=value1 class source*

where:

| | |
|---|---|
| **rrr** | = severity |
| **mmm** | = message describing the event |
| **slot1** | = slot (or field) in the event |
| **value1** | = value assigned to the slot |
| **class** | = type of event |
| **source** | = adapter (or program) that generated the event |

For example, to simulate an NFS_No_Response event generated by the LOGFILE adapter on a workstation where the hostname is *myhost* and the server name is *myserver*, enter the following command:

**wpostemsg -r FATAL -m ″Test Message″ hostname=myhost server=myserver NFS_No_Response LOGFILE**

See the *TME 10 Enterprise Console User's Guide* for more information about the **wpostemsg** command.

## Trouble Ticket

When viewing an event group or source in TEC, the "Event" menu item contains an action called "Trouble Ticket". When this action is selected, TEC tries to run a program called TroubleTicket.sh in the **$BINDIR\TME®\TEC** directory. When you install the TEC Integration Facility, a new TroubleTicket.sh is created in the **$BINDIR\INFOTEC** directory. This program performs the same functions as the Create_Record task. If you want TEC to run this program when you select ″Trouble Ticket″ from the Event menu, copy TroubleTicket.sh from the **$BINDIR\INFOTEC** directory to the **$BINDIR\TME\TEC** directory.

## Creating a Problem Record

The TEC Integration Facility provides four ways to create a problem record. You can:

- Select **Trouble Ticket** from the Event menu while viewing a TEC event.
- Manually run the Create_Record task from the INFOTEC library while viewing a TEC event, by selecting **Execute on Selected Event** from the Task menu.
- Automatically run the Create_Record task from the INFOTEC library when TEC receives an event, by selecting **New with Selected Event** from the Automated Tasks menu.
- Write rules that run the Create_Record task from the INFOTEC library when TEC receives an event.

For more information about running tasks, see the *TME 10 Enterprise Console User's Guide*. For more information about writing rules, see the *Tivoli Enterprise Console Rule Builder's Guide*.

## Updating a Problem Record

The TEC Integration Facility provides three ways to update a problem record that was created previously using either the Create_Record task or the Trouble Ticket task. You can:

- Manually run the Update_Record task from the INFOTEC library while viewing a TEC event, by selecting **Execute on Selected Event** from the Task menu.
- Automatically run the Update_Record task from the INFOTEC library when TEC receives an event, by selecting **New with Selected Event** from the Automated Tasks menu.
- Write rules that run the Update_Record task from the INFOTEC library when TEC receives an event.

When the Update_Record task updates a problem record, it updates all the fields listed in the Problem Service configuration file. It is recommended that you do not log on to Tivoli Information Management for z/OS and manually update these fields in a problem record because the Update_Record task will overlay your changes the next time the task runs.

## Deleting a Problem Record

The TEC Integration Facility provides three ways to delete a problem record that was created previously using either the Create_Record task or the Trouble Ticket task. You can:

- Manually run the Delete_Record task from the INFOTEC library while viewing a TEC event, by selecting **Execute on Selected Event** from the Task menu.
- Automatically run the Delete_Record task from the INFOTEC library when TEC receives an event, by selecting **New with Selected Event** from the Automated Tasks menu.
- Write rules that run the Delete_Record task from the INFOTEC library when TEC receives an event.

## Task Status

There are different ways to view the status of a task depending on how you run the task. For the Trouble Ticket task, a message box is displayed when the task finishes. If you manually run a task from the INFOTEC library, a message box is displayed when the task finishes. If you automatically run a task from the INFOTEC library, no message box is displayed when the task finishes. If you run a task from a rule, you can view the task's status by selecting **View Action Status** while viewing an event.

Additional sources of information are the Problem Service log (infogw.log) and the HLAPI client log (idblog.act).

## Rules

Rules enable you to specify what actions occur automatically when events that meet certain conditions are received by TEC. The TEC Integration Facility provides you with a sample rule set in the **$BINDIR\INFOTEC** directory. This rule set performs the following actions:

- If a new NFS_No_Response event is received, the Create_Record task is run to create a problem record in the Tivoli Information Management for z/OS database.
- If a TEC administrator acknowledges the event, the Update_Record task is run to update the corresponding problem record.
- If a TEC administrator closes the event, the Update_Record task is run to update the corresponding problem record.
- If a duplicate NFS_No_Response event is received within 10 minutes of a previous NFS_No_Response event, the repeat count in the previous event is incremented, the duplicate event is discarded, and the Update_Record task is run to update the corresponding problem record that was created by the previous event.
- If an NFS_OK event is received within 10 minutes of a previous NFS_No_Response event, the Delete_Record task is run to delete the corresponding problem record that was created by the previous NFS_No_Response event.

You can use the sample rule set as a base for writing a rule set that performs the processing needed by your organization. Before using the sample rule set, you must edit the infotec.rls file and change all instances of ″??????″ to the host name of the workstation on which the INFOTEC task library is installed.

To add the rule set to your rule base, perform the following steps from the command prompt on the workstation that is running the TEC Server:

1. Enter **wimprbrules** to import the rule set into the rule base
2. Enter **wcompbrules** to compile the rule base
3. Enter **wloadrb** to load the rule base
4. Enter **wstopesvr** to stop the TEC Event Server
5. Enter **wstartesvr** to start the TEC Event Server

See the *Tivoli Enterprise Console Rule Builder's Guide* for more information about writing and using rules. See the *TME 10 Enterprise Console User's Guide* for more information about commands that can be issued at the command prompt.

## Mapping Event Data to Problem Records

The TEC Integration Facility uses the Problem Service configuration file to map event data to problem record fields. A sample configuration file is in the **$BINDIR\INFOTEC** directory. The following list shows some sample data mappings:

```
Event data  -->  maps to  --> Problem Record field
----------                    --------------------
Severity                      Current Priority
  FATAL         --->             1
  CRITICAL      --->             2
  MINOR         --->             3
  WARNING       --->             3
  HARMLESS      --->             4
  UNKNOWN       --->             4
Status                        Problem Status
  OPEN          --->             INITIAL
  ACK           --->             OPEN
  CLOSED        --->             CLOSED
```

Each event has a unique event number that is generated by TEC. When either the Create_Record or Trouble Ticket task creates a problem record, it stores the event number in a field in the problem record. The sample configuration file uses the **TEC EVENT ID** field (s-word index S14FC). When either the Update_Record or Delete_Record task is run for an event, it uses the event number to search for the corresponding record in the Tivoli Information Management for z/OS database. Therefore, the Update_Record and Delete_Record tasks can only process problem records that were created by a task, such as the Create_Record task or the Trouble Ticket task, that stores the event number in the **TEC EVENT ID** field. Make sure that the TEC EVENT ID field is included in the PIDTs or data views that you specified in your Problem Service configuration file. See "Preparing the HLAPI Data Views on MVS" on page 293 for additional information. However, you can modify both the TEC Integration Facility tasks and the Problem Service configuration file to implement your own methodology for relating TEC events to Tivoli Information Management for z/OS problem records.

You can modify the Problem Service configuration file to change the data that is passed from an event to a problem record. If you change the configuration file, you may need to change the scripts invoked by the TEC Integration Facility tasks. Before using the sample configuration file, you must edit the blmygc.cfg file and replace all occurrences of ″??????″ with the appropriate values for your system. See “Customizing Tasks” for more information about customizing the TEC Integration Facility tasks. See “Problem Service Installation” on page 261 and “Customizing Your Problem Service Configuration File” on page 271 for more information about installing and configuring Problem Service.

When you run the Update_Record task, Problem Service expects the task to pass all of the fields specified in the configuration file for the ″Update″ transaction. If the task omits one or more fields, Problem Service deletes the corresponding field in the Tivoli Information Management for z/OS problem record. If you want to change the Update_Record task so that it only passes changed data, you also need to set ″InputJustChangedData=yes″ in the Problem Service configuration file. See “Customizing Your Problem Service Configuration File” on page 271 for more information about changing the configuration file.

Different types of TEC events can contain different data. Support for different event types can be implemented by modifying the provided sample tasks or creating new tasks, and by modifying the data mappings in the Problem Service configuration file.

## Customizing Tasks

The tasks created by the TEC Integration Facility use the transfer, update, and search transactions provided by Problem Service. The tasks are written as shell scripts and can be customized for your use. You can also add new tasks to the INFOTEC task library. The shell scripts are stored in the **$BINDIR\INFOTEC** directory. The following is a list of tasks and their associated shell scripts:

| **Task** | **Shell Script** |
|---|---|
| **Create_Record** | itcreate.sh |
| **Update_Record** | itupdate.sh |
| **Delete_Record** | itdelete.sh |
| **Trouble Ticket** | TroubleTicket.sh |

If you modify TroubleTicket.sh, you must copy it to **$BINDIR\TME\TEC** in order to activate your changes. If you modify any of the other scripts, do the following to activate your changes:

1. Open the Policy Region.
2. Open the INFOTEC task library.
3. Select the task to be changed, then use the right mouse button to select **Edit Task**. The Edit Task window displays.
4. Under Platforms Supported, click **Generic** to uncheck it.

5. Under Platforms Supported, click **Generic** to check it again. The New Executable For Task window displays.
6. Type the host name of the workstation on which the script is stored, and type the full path of the script. For example, **D:\Tivoli\bin\w32-ix86\INFOTEC\itcreate.sh**.
7. Click **Set & Close** to close the New Executable For Task window, then click **Change & Close** to close the Edit Task window.

For more information about editing a task in a task library, see the *Tivoli Framework User's Guide*

If you modify a shell script, be aware that some problem record fields only accept alphanumeric and national characters. Because event data can contain other special characters like periods, underscores, and hyphens, be sure that either the Problem Service configuration file or your shell script translates the special characters to acceptable characters, if necessary.

The tasks in the INFOTEC library can be written as C programs instead of shell scripts. See "Problem Service Application Programming Information" on page 299 for more information about writing programs that use Problem Service.

**Example:**

The Create_Record task in itcreate.sh sends 9 fields to Problem Service to create a problem record via the ″transfer″ transaction. The call to InfoGW::transfer is shown below:

```
RNID= idlcall $OID InfoGW::transfer { 9  \
      {\"Originator\" \"$MYSOURCE\"} \
      {\"StartDate\" \"$MYDATE\"} \
      {\"Priority\" \"$severity\"} \
      {\"EventNum\" \"$ev_key\"} \
      {\"RepeatCount\" \"$repeat_count\"} \
      {\"ClassDesc\" \"$MYCLASS\"} \
      {\"Description\" \"$MYDESC\"} \
      {\"Detail\" \"$MYDETAIL\"} \
      {\"Status\" \"$status\"}} \"TECevent\"
```

Suppose your event contains a slot called ″server″, and you want to store the value of that slot in the **System Name** field (s-word index S0CA5) when a problem record is created.

First, add a new mapping to the Problem Service configuration file as shown:

```
// ----------------------------------------------------------------
// Define additional Transfer mappings.
// ----------------------------------------------------------------
Transactions=transfer;
S0CA5(8)<<Server;     // ADDED THIS LINE!
S0E0F(45)<<ClassDesc;
S1260<<"TGATEW1";    // Indicates this record was transferred by this
                     //   Problem Service gateway.
```

Next, add the slot to the call to InfoGW::transfer and increment the field count:

```
RNID= idlcall $OID InfoGW::transfer { 10 \    # CHANGED THIS LINE!
      {\"Server\" \"$server\"} \             # ADDED THIS LINE!
      {\"Originator\" \"$MYSOURCE\"} \
      {\"StartDate\" \"$MYDATE\"} \
      {\"Priority\" \"$severity\"} \
      {\"EventNum\" \"$ev_key\"} \
      {\"RepeatCount\" \"$repeat_count\"} \
      {\"ClassDesc\" \"$MYCLASS\"} \
      {\"Description\" \"$MYDESC\"} \
      {\"Detail\" \"$MYDETAIL\"} \
      {\"Status\" \"$status\"}} \"TECevent\"
```

If the ″server″ slot contains special characters like periods, underscores or hyphens, you may need to translate these special characters to characters that Tivoli Information Management for z/OS accepts. This translation can be performed by the Problem Service configuration file as shown:

```
// -------------------------------------------------------------
// Define additional Transfer mappings.
// For "Server", translate periods and hyphens to #
// -------------------------------------------------------------
Transactions=transfer;
S0CA5 (8)<<change(Server, ".", "#", "-", "#");   // ADDED THIS LINE!
S0E0F (45)<<ClassDesc;
S1260<<"TGATEW1";     // Indicates this record was transferred by this
                      //   Problem Service gateway.
```

Finally, update the task library as described in “Customizing Tasks” on page 392.

## List of Files

**$BINDIR\INFOTEC directory:**

| | |
|---|---|
| blmygc.cfg | sample configuration file for Problem Service |
| infotec.rls | sample rule set |
| itcreate.sh | shell script for Create_Record task |
| itcrtlib.sh | shell script to create INFOTEC task library |
| itdelete.sh | shell script for Delete_Record task |
| itlib.tll | task definitions used by itcrtlib.sh |
| itupdate.sh | shell script for Update_Record task |
| TroubleTicket.sh | shell script for Trouble Ticket task |

# 31 Integrating with Tivoli Software Distribution

Tivoli Software Distribution automates the software distribution process to clients and servers throughout the enterprise. It allows you to install and update applications and software in a coordinated, consistent manner across UNIX and PC platforms. Tivoli Software Distribution simplifies the software distribution process, enabling timely client/server application deployment. Software Distribution gives the administrator a centralized software management capability to add new applications, update existing software with newer versions, and synchronize software on distributed systems.

## Overview of the Interface to Tivoli Software Distribution

You can run Tivoli Software Distribution from within Tivoli Information Management for z/OS via the Change Management facility. The software to be distributed is defined in a Tivoli software distribution file package. (The definition and creation of a file package is described in the *Tivoli Software Distribution Reference Manual*.) The target machines, those machines to which the software is to be distributed, are defined as those which are subscribed to the software distribution profile manager containing the file package. A change request record is created; part of the data contained in this change request record is the name of the software file package that is to be distributed. Once the change request record has been approved, an event is generated and sent via TCP/IP to the Tivoli Enterprise Console (TEC). The TEC server receives this as an event, and associates this event with the appropriate rules for processing the event. The rules associated with the event invoke a PERL program that causes the Software Distribution process to distribute the file package named in the change request record to the designated machines.

## Installing the Components Used by the Interface

This section lists the steps for installing the components used by the interface for Tivoli Information Management for z/OS to Tivoli Software Distribution. Tivoli Information Management for z/OS relies on Tivoli Software Distribution and the Tivoli Enterprise Console. Before you install the host and workstation components for this function, ensure that Tivoli Software Distribution is installed and that a package file can be created, with target machines subscribed to it. For information on installing Tivoli Software Distribution, refer to the *Tivoli Software Distribution User’s Guide*. You must have also installed the Tivoli Event Integration Facility for OS/390 (5697–C74).

Installation involves loading both host and workstation components. Although it is not essential to install the host components before installing the workstation components, that is the recommended order.

## Installing the Host Components

Follow these steps to install the host portion of the interface from Tivoli Information Management for z/OS to Tivoli Software Distribution. The steps involve modifying the panel flow for your Change Management process to add the invocation of the interface at the appropriate point(s), tailoring the configuration file used by the Tivoli Information Management for z/OS TEC Event Adapter, and tailoring the JCL for a batch job that runs this event adapter. It is assumed that you have already installed Tivoli Information Management for z/OS Version 7.1.

1. Modify the panel flow for your Tivoli Information Management for z/OS Change Management process. If you use the shipped version of these panels, you can skip this step. For an example of the necessary modifications, refer to the shipped version of panel BLG0CU00 (for the collection of data such as a file package name and related items), and panel BLG0M500 (to initiate a software distribution request when all approvers have approved the change request record).
2. Tailor the Tivoli Information Management for z/OS TEC Event Adapter configuration file located in the OS/390 UNIX System Services HFS path named

   ```
   /usr/lpp/InfoMan/tec/blgtecad.conf
   ```

   This is blgtecad.conf as it is shipped from Tivoli. Modify this file to identify the target TEC server by HOSTNAME or by PORT number.

   ```
   #  --------------------------------------------------------------------------------
   #   Description:
   #   Tivoli Information Management for z/OS Message Event Adapter Configuration file
   #
   #  ================================================================================
   #
   ServerLocation=hostname
   #
   #   Default: None
   #   Description: Specifies the name of the host on which the event
   #   server is installed. The value in this field must be the
   #   hostname of the system where the TEC server
   #   is installed. The ServerLocation keyword is required.
   #   The ServerLocation keyword may contain more than one value,
   #   separated by commas. If more than one value is specified, the
   #   first location for which a connection can be established will
   #   be the location used.
   #
   #ServerPort=0
   #
   #   Default: 0
   #   Description: Specifies the port number on which the event
   #     server listens. This value should be 0 unless the portmapper
   #     is not available on the server. If the value is not specified
   #     or is specified as 0, the port number is retrieved by calling
   #     the portmapper. The ServerPort keyword may contain more than
   #     one value, separated by commas. If more than one port number
   #     is specified, each port number specified should be paired
   #     with a ServerLocation specified.
   #
   #TestMode=yes
   #
   #   Default: None
   #   Valid values: yes|YES
   #   Description: When TestMode is specified as yes or YES, events
   #     are not sent to the TEC console but instead flushed to
   #     a text file with a name that matches the value specified for
   #     serverlocation.
   ```

```
#
#
BufferEvents=no
#
```

3. Tailor the JCL for the batch job used to run the Tivoli Information Management for z/OS TEC Event Adapter program. Sample JCL for this job can be found in member BLGTECAD in the SBLMSAMP sample library. Instructions that describe the required modifications are included.

   **Note:** This job will require RACF authority to run program blgtecad which is installed in the OS/390 UNIX System Services HFS file named

   ```
   /usr/lpp/InfoMan/tec/blgtecad
   ```

   This is the JCL for job BLGTECAD as it is shipped from Tivoli:

   ```
   //BLGTECAD JOB
   //*-------------------------------------------------------------------
   //*-------------------------------------------------------------------
   //* This is a sample JCL stream for running the Tivoli Information
   //* Management for z/OS TEC adapter TSX.  This job should be scheduled
   //* to run at intervals appropriate for your installation.  You must
   //* change the dataset names for DD names STEPLIB, ISPPROF, ISPPLIB,
   //* ISPMLIB, ISPSLIB, ISPTLIB, BLGTRACE, ISPLLIB, SYSPROC, and
   //* SYSTSPRT to correspond to the data set names at your installation.
   //* You may also need to change the privilege class and session member
   //* listed in the parameters below.  The privilege class used must
   //* have the authority to search for and update CHANGE MANAGEMENT
   //* records.
   //*
   //* The Tivoli Information Management for z/OS "RDR" operator command
   //* can be used to terminate this job,
   //* E.G. "F procname,RDR,FLUSH=BLGTECAD"
   //* where procname designates the BLX-SP this job is connected to.
   //*
   //* ---
   //*
   //* DD names ISPxxxx are the libraries required by ISPF.
   //* DD name BLGTRACE can be used to trace the flow of TSPs
   //*    for informational or debug purposes.  Refer to the
   //*    TERMINAL SIMULATOR GUIDE AND REFERENCE for information
   //*    on using "TRACE" with TSPs.
   //* DD name SYSTSPRT is for TSO background output, which is where
   //*    generated messages will be written.
   //*

   //* USERID is the MVS user ID assigned to this job.
   //*   This ID should be in the privilege class specified in the
   //*   parameters below.
   //*--------------------------------------------------------------------
   //STEP1    EXEC PGM=IKJEFT01,DYNAMNBR=25,REGION=4M,TIME=1440
   //STEPLIB  DD   DISP=SHR,DSN=ISP.SISPLPA
   //         DD   DISP=SHR,DSN=ISP.SISPLOAD
   //         DD   DISP=SHR,DSN=BLM.SBLMMOD1
   //ISPPROF  DD   DISP=SHR,DSN=USERID.ISPF.ISPPROF
   //ISPPLIB  DD   DISP=SHR,DSN=ISP.SISPPXXX
   //ISPTLIB  DD   DISP=SHR,DSN=ISP.SISPTXXX
   //ISPMLIB  DD   DISP=SHR,DSN=ISP.SISPMXXX
   //ISPSLIB  DD   DISP=SHR,DSN=ISP.SISPSLIB
   //ISPLLIB  DD   DISP=SHR,DSN=ISP.SISPLPA
   //         DD   DISP=SHR,DSN=ISP.SISPLOAD
   //         DD   DISP=SHR,DSN=BLM.SBLMMOD1
   //SYSPROC  DD   DISP=SHR,DSN=ISP.SISPCLIB
   //BLGTSX   DD   DISP=SHR,DSN=BLM.SBLMTSX
   //SYSPRINT DD   SYSOUT=A
   ```

```
//BLGTRACE DD   SYSOUT=A
//SYSTSPRT DD   SYSOUT=A,DCB=(RECFM=VBA,LRECL=125,BLKSIZE=0)
//SYSTSIN  DD   *
  PROFILE PREFIX(USERID)
  ISPSTART PGM(BLGINIT) +
  PARM(IRC(RUN BLGTAGSD,;QUIT) CLASS(MASTER) SESS(00))
/*
//
```

**Note:** If you are using a database that is logically partitioned, you should choose a privilege class that has access to all partitions so that this job can access any record in any database partition. The *Tivoli Information Management for z/OS Program Administration Guide and Reference* contains additional information on logically partitioned databases.

4. If the Tivoli Information Management for z/OS TEC Event Adapter is installed in a directory path other than the default, TSX BLGTAGSD must be modified to use the correct directory path.

When you have completed these steps, you can proceed with installing the workstation components for this interface.

## Installing the TEC Components

Before beginning this portion of the installation, ensure that the HLAPI client can communicate with Tivoli Information Management for z/OS on MVS.

This portion of the installation should be done by the TEC administrator at your site.

- Create a directory for the TEC components.
- Insert the Tivoli Information Management for z/OS installation CD-ROM into your CD-ROM drive and copy **/Tivoli_Int/SWD/info_swd.tar** into the directory you created in the previous step.
- Unpack the tar file by typing **tar -xvf info_swd.tar**. To run tar on Windows NT, you must set the Tivoli environment variables and start the bash shell.

## Software Distribution from TEC

There are three parts to distribute software from TEC events. The first is to define the classes; this is described in “Class Definitions for TEC Events”. Within TEC there are rules that identify the events. The rules that identify the Tivoli Information Management for z/OS event are described in “Rules for Processing Events” on page 399. The program that initiates the actual software distribution is described in “PERL Program To Initiate Distribution of a File Package” on page 400.

### Class Definitions for TEC Events

These are the class definitions for TEC events that can be used to initiate a distribution of software:

```
#
#   INFO.baroc
#
#  Example for events
#
#

TEC_CLASS:
        INFO_sw_dist ISA EVENT
                DEFINES {
                        severity:              default = HARMLESS;
                        filemode:              STRING, default="a";
                        distmode:              STRING, default="b";
                        fp_name:               STRING;
                        subscribers:           STRING;
                        };
END

TEC_CLASS:
        INFO_close_event ISA EVENT
                DEFINES {
                        severity:              default = WARNING;
                        close_ev_class:        STRING;
                        close_ev_handle:       INTEGER;
                        close_sv_handle:       INTEGER;
                        close_date_reception:  INT32;
                        };
END
```

## Rules for Processing Events

The class `INFOsw_dist`, when evaluated by the appropriate rules, can be used to trigger the distribution of a file package. To include these event definitions, use the following steps and practices:

1. Check the class by running `wchkclass INFO.baroc rbname` where `rbname` is the name of the rulebase to which you intend to include these classes.
2. Import the class by running `wimprbclass INFO.baroc rbname` where `rbname` is the name of the rulebase to which you wish to import the definitions.
3. Load the rulebase by running `wloadrb rbname`.
4. Stop and restart the event server.

The following rules process an `INFO_sw_dist` event and call a task to distribute a file package. The first rule is to call a PERL program that triggers a software distribution.

```
/*                                                                    */
/*                                                                    */

rule: distribute_Package:  (
       event: _ev of_class 'INFO_sw_dist'
               where [
                      severity: equals  'HARMLESS'
                      ],

       reception_action: (
               set_event_status( _ev, 'ACK' )
               ),

       reception_action: (
               exec_program( _ev,
                             'perl SWDist.pl',
                             'xxx' ,
                             [ ],
                             'YES')
                )

  ).
```

**Note:** If you encounter difficulty with this program, the most likely cause is the `'perl SWDist.pl'` statement. The PERL interpreter is shipped with TEC. One possible cause of the problem is that the PERL interpreter is not pointed to correctly; ensure that the path is correct. It is also possible that there are path problems getting to `SWDist.pl`. You may need to change this statement to directly point to the PERL interpreter and/or `SWDist.pl`.

The second rule (`close_cause_event`) can be used to close an event when its corresponding ticket in Tivoli Information Management for z/OS is closed. This rule assumes that the ticket has captured the message ID, as well as the class name of the event that had the ticket opened. Usually these events are related to a Tivoli Information Management for z/OS record. In many instances, it was the event that caused the Tivoli Information Management for z/OS record to be created.

```
rule: close_cause_event:    (

       event: _ev of_class 'INFO_close_event'
               where [
  severity: equals 'WARNING',
                      msg: _msg,
                      close_ev_class:  _cclass,        /* class of event that opened ticket  */
                      close_ev_handle: _cev,           /* event_handle of causing event      */
                      close_sv_handle: _csv,           /* server_handle of causing event     */
                      close_date_reception:  _cdate    /* date of causing event              */
  ],

         action: close_original_event:(
       all_instances(
               event: _ev_close of_class _cclass
               where [
                       event_handle:   equals _cev,
                       server_handle:  equals _csv,
                       date_reception: equals _cdate
                     ]
                     ),

                set_event_status( _ev_close, 'CLOSED')
                ),

       reception_action: (
               set_event_status( _ev, 'ACK' )
               )

  ).
```

## PERL Program To Initiate Distribution of a File Package

This code demonstrates how to initiate a software distribtion using PERL code.

```
# SWDist.pl
#
#
#   Perl program, invoked via rules from TEC to initiate a S/W distribution
#

# This captures the filemode slot
   $filemode = $ENV{'filemode'};
   if ( !$name ) { $filemode = "a"; }

# This captures the distmode
   $distmode = $ENV{'distmode'};
   if ( !$distmode ) { $distmode = "b"; }

# This captures the file package name from the message
   $fp_name  = $ENV{'fp_name'};
   if ( !$fp_name ) {
     print "No file package name was specified! Exit process\n";
     exit (2);
#     $fp_name = "@Sample_Files";
   }

# This captures the subscriber's list, if present
   $sublist  = $ENV{'sublist'};



#
#  Now, issue the distribute command and test the return code.
#

# First, build the command we wish to issue
   $Command = "wdistfp -".$filemode." -".$distmode." ".$fp_name." ".$sublist ;

#   print "Command is = ".$Command."\n";

   $rc = system( $Command );
#   printf  "wdistfp return code = %1u \n", $rc/256;
   exit $rc/256;                    # Give the wdistfp return code.
```

## Using the Interface to Tivoli Software Distribution

Using this interface consists of two steps: creating a change request record and approving the change request record. Remember, the sample JCL for running the Tivoli Information Management for z/OS TEC Event Adapter is contained in BLGTECAD, shipped in the SBLMSAMP sample library. It must be modified for your installation. Once this JCL is started, it can be stopped with the RDR command, described in the *Tivoli Information Management for z/OS Operation and Maintenance Reference* document.

### Example of Distributing the Tivoli Information Management for z/OS HLAPI Clients

Tivoli Information Management for z/OS provides a set of Software Distribution file packages that enable you to distribute and install the Tivoli Information Management for z/OS HLAPI clients. The file packages provide “containers” for, or the mechanism to distribute, the HLAPI clients. The file packages contain references to the files and directories to be distributed, and options or directions on how to distribute them. To install the file packages, look at the READ.ME file in the directory **\HLAPI\SW_DIST** on the installation CD.

The file packages make use of configuration scripts and the “unattended” install feature of the HLAPI clients in order to automate the installation process. The file packages, configuration scripts, and installation response files can all be customized and should be customized to fit your particular installation. For more information on the installation response files provided with the HLAPI clients, see the *Tivoli Information Management for z/OS Client Installation and User’s Guide*.

The following Tivoli Information Management for z/OS HLAPI client features can be installed using Tivoli Software Distribution:

HLAPI/2
HLAPI/NT
HLAPI/AIX
HLAPI/HP
HLAPI/Solaris

HLAPI/CICS® and HLAPI/USS are installed using SMP/E.

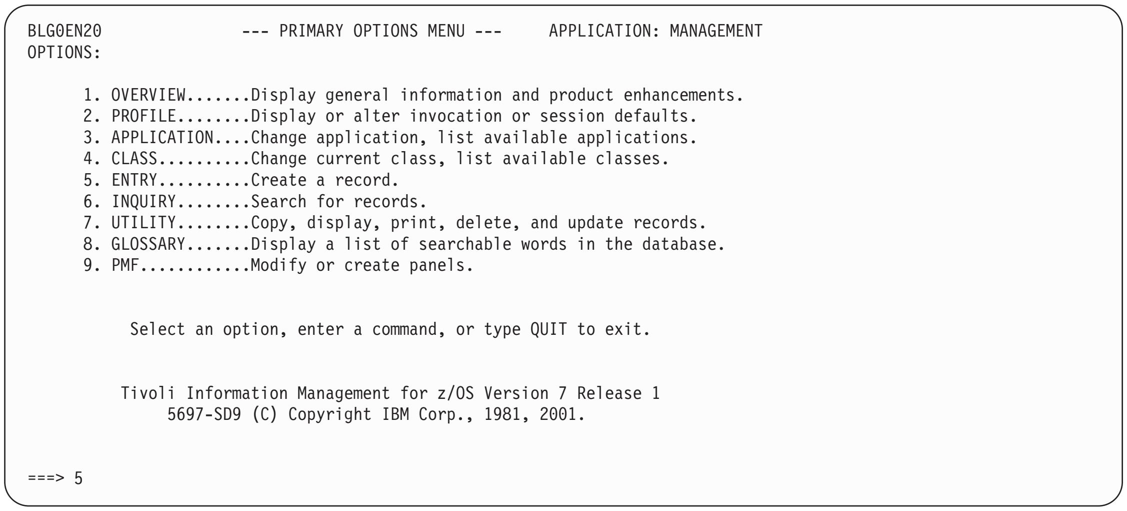
To distribute software upon approval of a Tivoli Information Management for z/OS change request record, you need:

- Tivoli Software Distribution Version 3.1
- Tivoli Enterprise Console Version 3.1

## Creating a Change Request Record

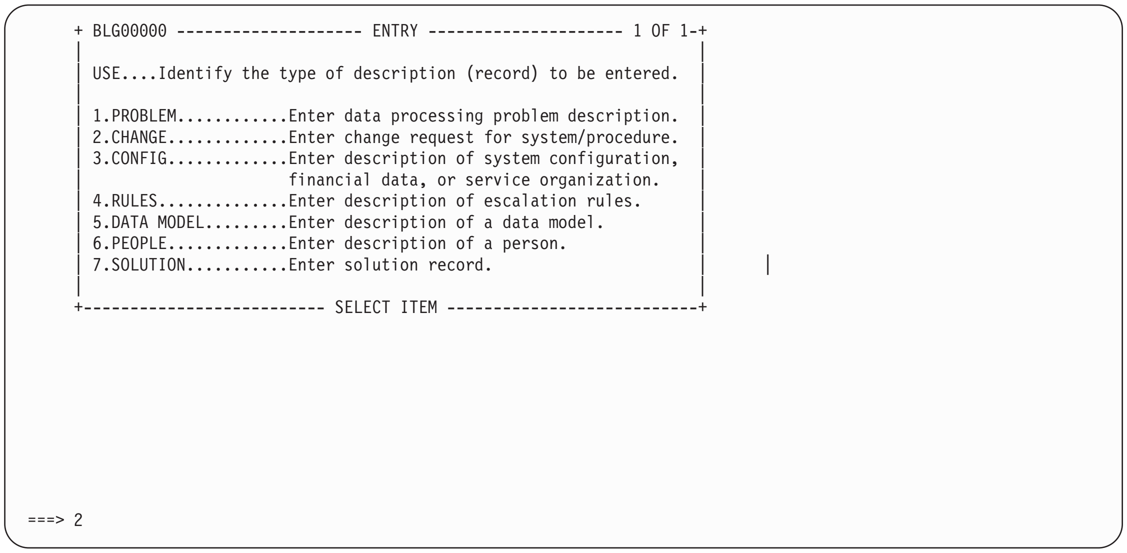
In this stage of the process, you must create a change request record that defines the change that is to occur; in this instance, the software that is to be installed. A file package has previously been defined that identifies the software to be distributed.

From the initial Tivoli Information Management for z/OS screen, type 5 and press Enter.



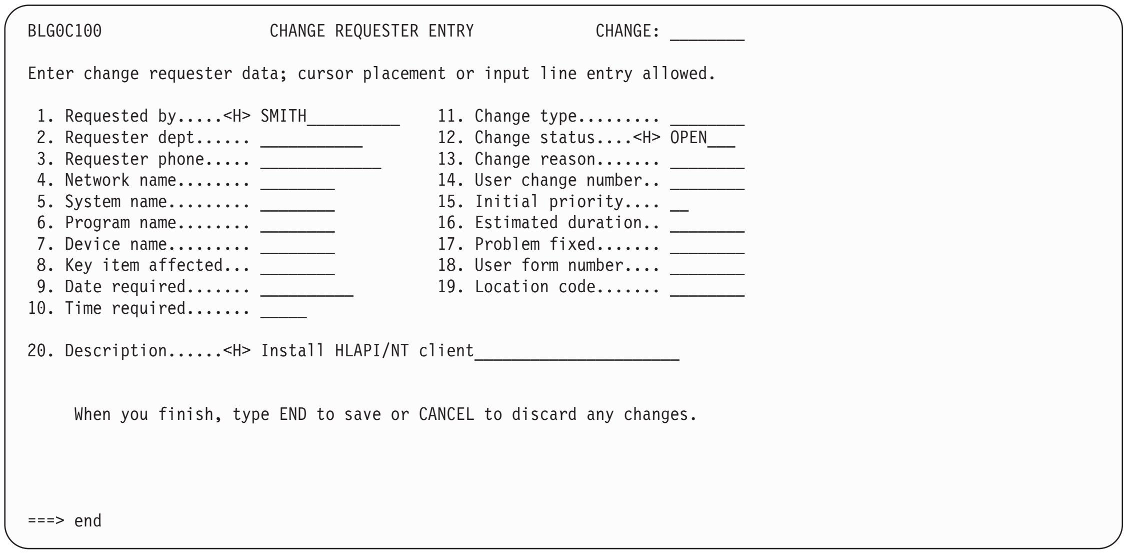
```
BLG0EN20              --- PRIMARY OPTIONS MENU ---     APPLICATION: MANAGEMENT
OPTIONS:

      1. OVERVIEW.......Display general information and product enhancements.
      2. PROFILE........Display or alter invocation or session defaults.
      3. APPLICATION....Change application, list available applications.
      4. CLASS..........Change current class, list available classes.
      5. ENTRY..........Create a record.
      6. INQUIRY........Search for records.
      7. UTILITY........Copy, display, print, delete, and update records.
      8. GLOSSARY.......Display a list of searchable words in the database.
      9. PMF............Modify or create panels.


           Select an option, enter a command, or type QUIT to exit.


          Tivoli Information Management for z/OS Version 7 Release 1
               5697-SD9 (C) Copyright IBM Corp., 1981, 2001.


===> 5
```

Next, type 2 and press Enter to identify this record as a change request record.



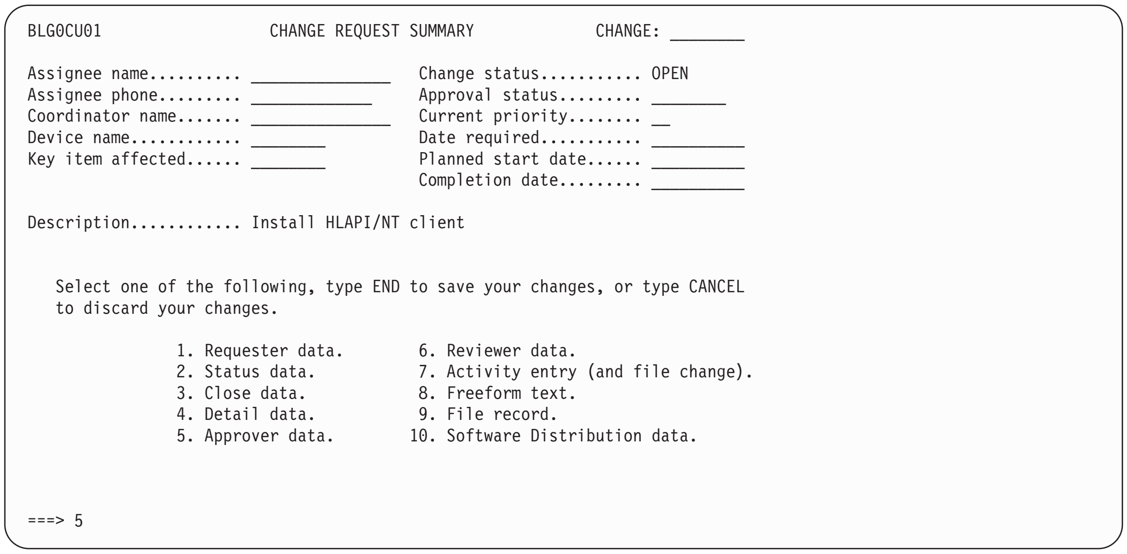
```
     + BLG00000 -------------------- ENTRY --------------------- 1 OF 1-+
     |                                                                  |
     | USE....Identify the type of description (record) to be entered.  |
     |                                                                  |
     | 1.PROBLEM...........Enter data processing problem description.   |
     | 2.CHANGE............Enter change request for system/procedure.   |
     | 3.CONFIG............Enter description of system configuration,   |
     |                     financial data, or service organization.     |
     | 4.RULES.............Enter description of escalation rules.       |
     | 5.DATA MODEL.........Enter description of a data model.          |
     | 6.PEOPLE............Enter description of a person.               |
     | 7.SOLUTION..........Enter solution record.                       |      |
     |                                                                  |
     +------------------------- SELECT ITEM --------------------------+








===> 2
```

Complete the required fields (the name of the requester, the change status, and a description of the change request) and any other fields as needed. In this example, a software distribution file package called `ntfp.fp` will be used.



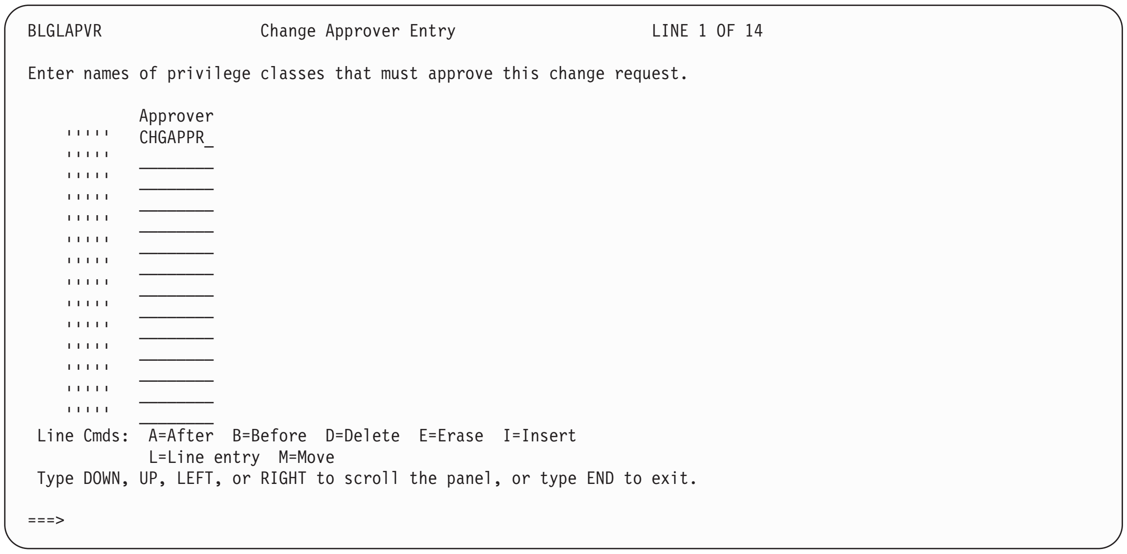
```
BLG0C100                CHANGE REQUESTER ENTRY            CHANGE: ________

Enter change requester data; cursor placement or input line entry allowed.

 1. Requested by.....<H> SMITH__________    11. Change type......... ________
 2. Requester dept...... ___________        12. Change status....<H> OPEN___
 3. Requester phone..... _____________      13. Change reason....... ________
 4. Network name........ ________           14. User change number.. ________
 5. System name......... ________           15. Initial priority.... __
 6. Program name........ ________           16. Estimated duration.. ________
 7. Device name......... ________           17. Problem fixed....... ________
 8. Key item affected... ________           18. User form number.... ________
 9. Date required....... __________         19. Location code....... ________
10. Time required....... _____

20. Description......<H> Install HLAPI/NT client______________________


     When you finish, type END to save or CANCEL to discard any changes.




===> end
```

A summary of the change request is displayed. Type **5** and press Enter to identify the approver(s) for this change request.



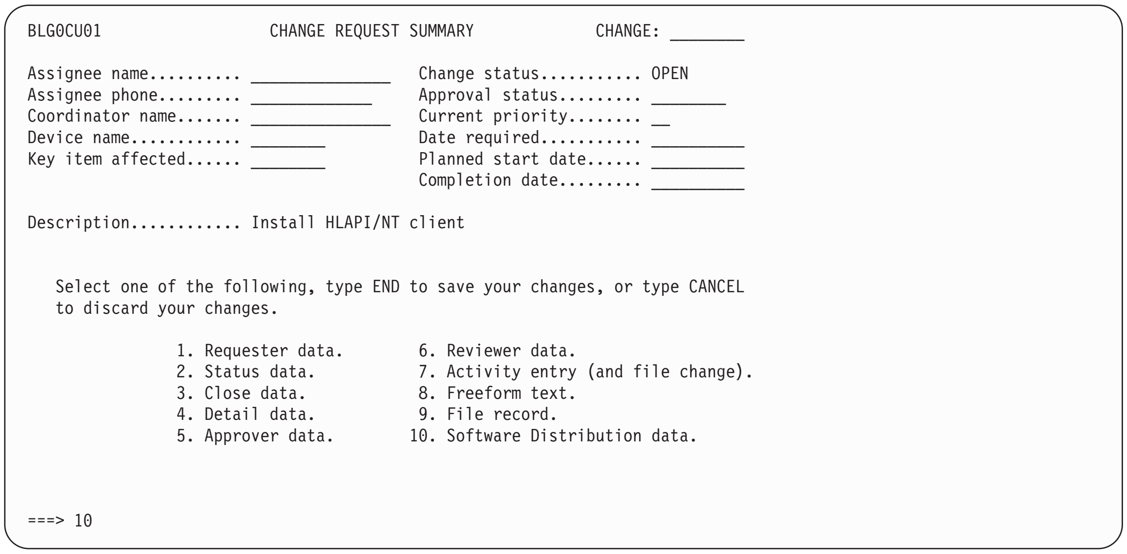
```
BLG0CU01                CHANGE REQUEST SUMMARY            CHANGE: ________

Assignee name.......... ________________   Change status........... OPEN
Assignee phone......... ______________     Approval status......... ________
Coordinator name....... ________________   Current priority........ __
Device name............ ________           Date required........... __________
Key item affected...... ________           Planned start date...... __________
                                           Completion date......... __________

Description............ Install HLAPI/NT client


   Select one of the following, type END to save your changes, or type CANCEL
   to discard your changes.

                1. Requester data.        6. Reviewer data.
                2. Status data.           7. Activity entry (and file change).
                3. Close data.            8. Freeform text.
                4. Detail data.           9. File record.
                5. Approver data.        10. Software Distribution data.



===> 5
```

Identify the privilege class of the approver. In this example, assume that a CHGAPPR privilege class has been created for the purpose of overseeing change requests. On panel BLGLAPVR, the privilege class CHGAPPR is entered; any users belonging to that privilege class are able to approve the change request.



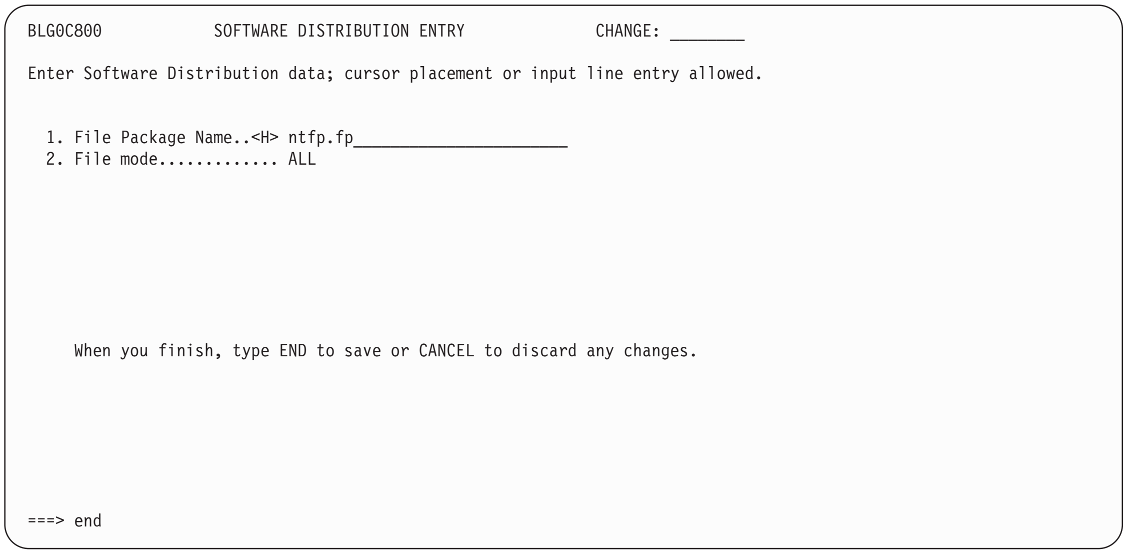
```
BLGLAPVR                Change Approver Entry                    LINE 1 OF 14

Enter names of privilege classes that must approve this change request.

            Approver
    '''''   CHGAPPR_
    '''''   ________
    '''''   ________
    '''''   ________
    '''''   ________
    '''''   ________
    '''''   ________
    '''''   ________
    '''''   ________
    '''''   ________
    '''''   ________
    '''''   ________
    '''''   ________
    '''''   ________
 Line Cmds:  A=After  B=Before  D=Delete  E=Erase  I=Insert
             L=Line entry  M=Move
 Type DOWN, UP, LEFT, or RIGHT to scroll the panel, or type END to exit.

===>
```

The next task is to identify the name of the file package that contains the software. Type 10 and press Enter.

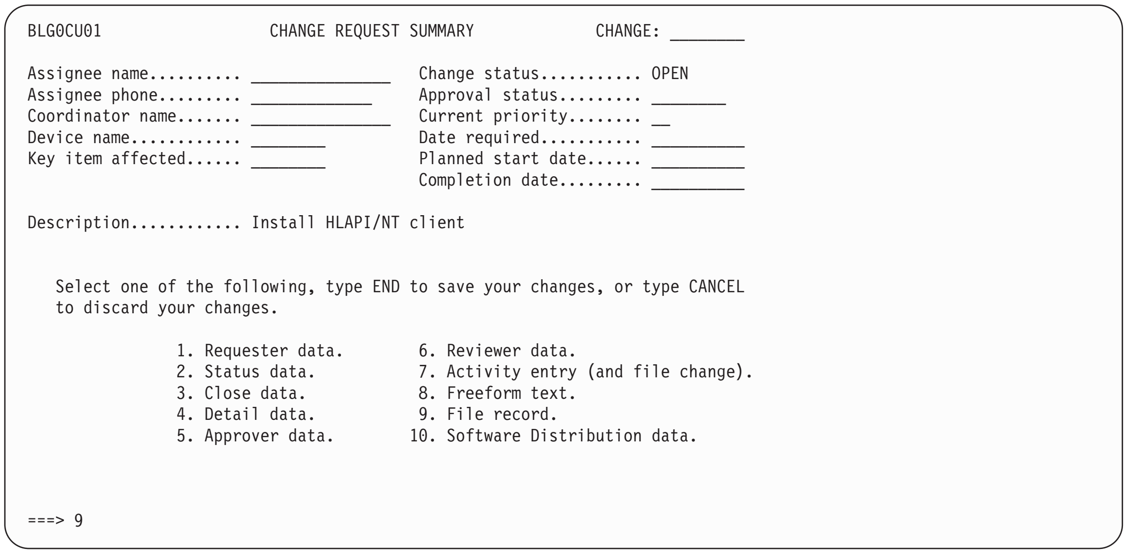


```
BLG0CU01                 CHANGE REQUEST SUMMARY             CHANGE: ________

Assignee name.......... _______________   Change status........... OPEN
Assignee phone......... _____________     Approval status......... ________
Coordinator name....... _______________   Current priority........ __
Device name............ ________          Date required........... __________
Key item affected...... ________          Planned start date...... __________
                                          Completion date......... __________

Description............ Install HLAPI/NT client


   Select one of the following, type END to save your changes, or type CANCEL
   to discard your changes.

                1. Requester data.        6. Reviewer data.
                2. Status data.           7. Activity entry (and file change).
                3. Close data.            8. Freeform text.
                4. Detail data.           9. File record.
                5. Approver data.        10. Software Distribution data.



===> 10
```

Enter the file package name. In the File mode field, you can type ALL if all files in the package are to be distributed, MOD if only modified files are to be distributed, or NEW if only files modified since the last distribution are to be distributed.

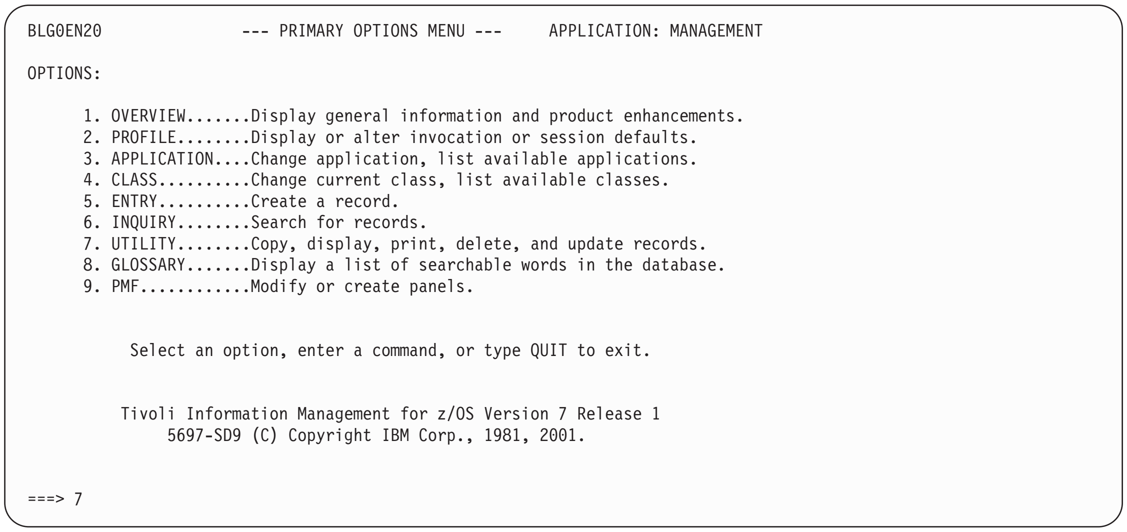

```
BLG0C800           SOFTWARE DISTRIBUTION ENTRY             CHANGE: ________

Enter Software Distribution data; cursor placement or input line entry allowed.


  1. File Package Name..<H> ntfp.fp_______________________
  2. File mode............ ALL







     When you finish, type END to save or CANCEL to discard any changes.








===> end
```

Type 9 and press Enter to file the record.


```
BLG0CU01                 CHANGE REQUEST SUMMARY            CHANGE: ________

Assignee name.......... _______________   Change status........... OPEN
Assignee phone......... _____________     Approval status......... ________
Coordinator name....... _______________   Current priority........ __
Device name............ ________          Date required........... __________
Key item affected...... ________          Planned start date...... __________
                                          Completion date......... __________

Description............ Install HLAPI/NT client


   Select one of the following, type END to save your changes, or type CANCEL
   to discard your changes.

                1. Requester data.        6. Reviewer data.
                2. Status data.           7. Activity entry (and file change).
                3. Close data.            8. Freeform text.
                4. Detail data.           9. File record.
                5. Approver data.        10. Software Distribution data.



===> 9
```

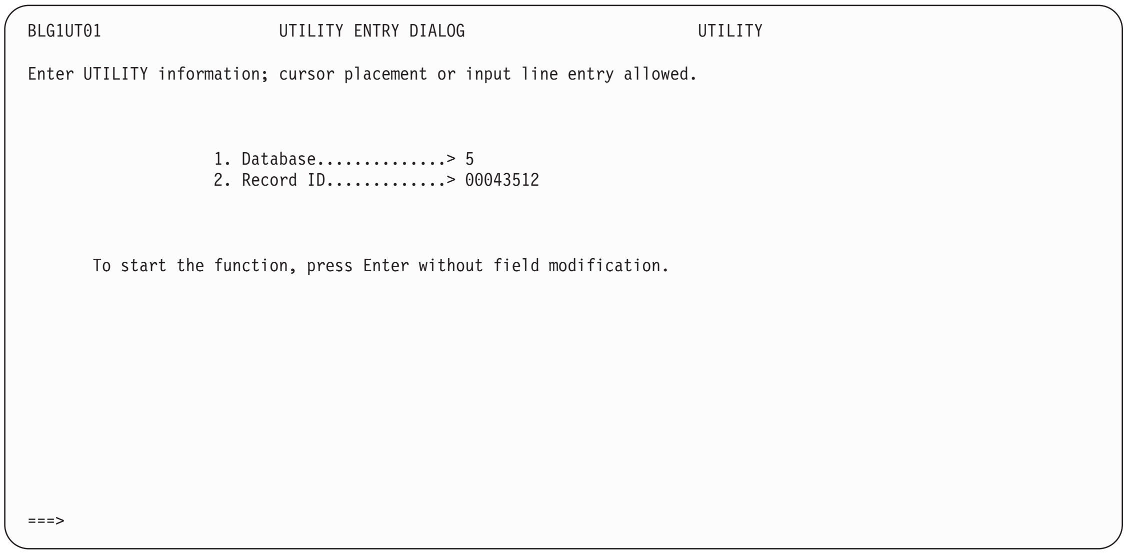
## Approving the Change Request Record

The next steps permit you to accept or reject approval of the change request. From BLG0EN20, type 7 and press Enter to begin the approval process.



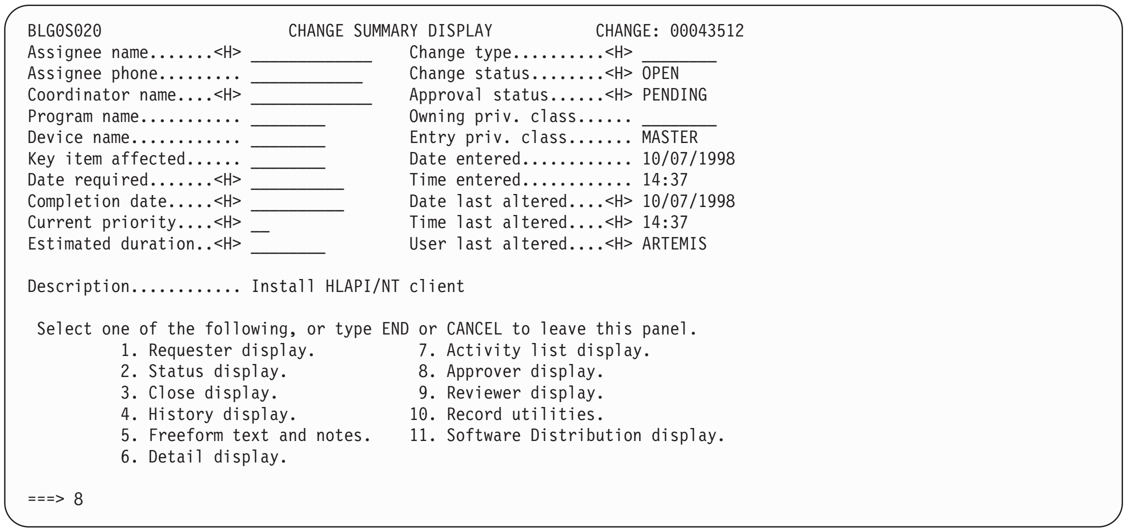
```
BLG0EN20              --- PRIMARY OPTIONS MENU ---     APPLICATION: MANAGEMENT

OPTIONS:

      1. OVERVIEW.......Display general information and product enhancements.
      2. PROFILE........Display or alter invocation or session defaults.
      3. APPLICATION....Change application, list available applications.
      4. CLASS..........Change current class, list available classes.
      5. ENTRY..........Create a record.
      6. INQUIRY........Search for records.
      7. UTILITY........Copy, display, print, delete, and update records.
      8. GLOSSARY.......Display a list of searchable words in the database.
      9. PMF............Modify or create panels.


           Select an option, enter a command, or type QUIT to exit.


          Tivoli Information Management for z/OS Version 7 Release 1
               5697-SD9 (C) Copyright IBM Corp., 1981, 2001.


===> 7
```

Display the record that was previously created.



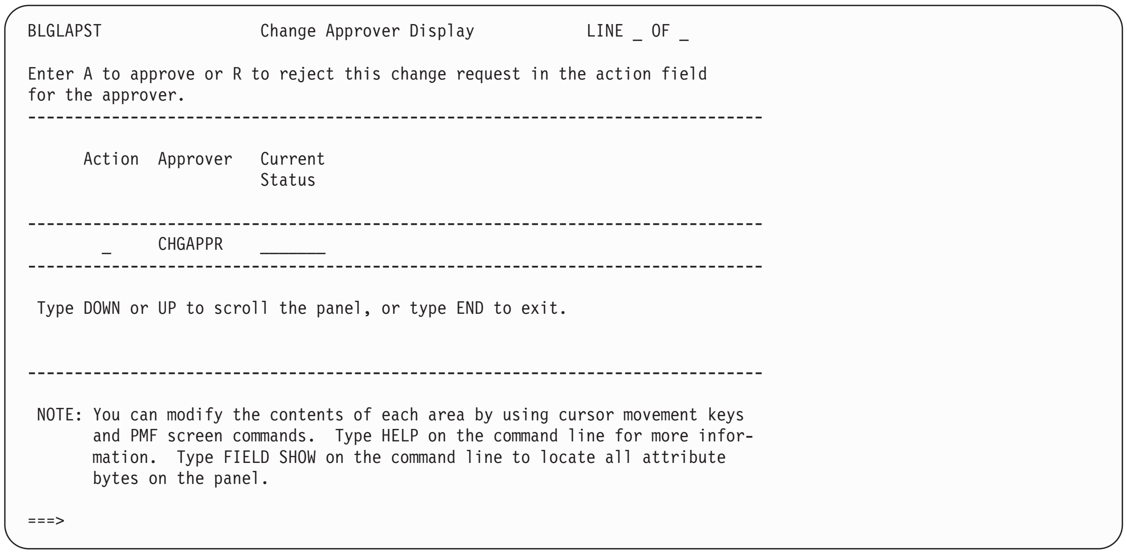
```
BLG1UT01                 UTILITY ENTRY DIALOG                         UTILITY

Enter UTILITY information; cursor placement or input line entry allowed.



                    1. Database.............> 5
                    2. Record ID............> 00043512



       To start the function, press Enter without field modification.











===>
```

From the Change Summary Display, type 8 and press Enter.



```
BLG0S020                  CHANGE SUMMARY DISPLAY          CHANGE: 00043512
Assignee name.......<H> _____________   Change type..........<H> ________
Assignee phone......... ____________    Change status........<H> OPEN
Coordinator name....<H> _____________   Approval status......<H> PENDING
Program name........... ________        Owning priv. class...... ________
Device name............ ________        Entry priv. class....... MASTER
Key item affected...... ________        Date entered............ 10/07/1998
Date required.......<H> __________      Time entered............ 14:37
Completion date.....<H> __________      Date last altered....<H> 10/07/1998
Current priority....<H> __              Time last altered....<H> 14:37
Estimated duration..<H> ________        User last altered....<H> ARTEMIS

Description............ Install HLAPI/NT client

 Select one of the following, or type END or CANCEL to leave this panel.
          1. Requester display.          7. Activity list display.
          2. Status display.             8. Approver display.
          3. Close display.              9. Reviewer display.
          4. History display.           10. Record utilities.
          5. Freeform text and notes.   11. Software Distribution display.
          6. Detail display.

===> 8
```

At the Change Approver Display, the authorized approver(s) can elect to accept the change or reject the change.



```
BLGLAPST                Change Approver Display           LINE _ OF _

Enter A to approve or R to reject this change request in the action field
for the approver.
-------------------------------------------------------------------------------

      Action  Approver   Current
                         Status

-------------------------------------------------------------------------------
        _     CHGAPPR    _______
-------------------------------------------------------------------------------

 Type DOWN or UP to scroll the panel, or type END to exit.


-------------------------------------------------------------------------------

 NOTE: You can modify the contents of each area by using cursor movement keys
       and PMF screen commands.  Type HELP on the command line for more infor-
       mation.  Type FIELD SHOW on the command line to locate all attribute
       bytes on the panel.

===>
```

Once the change request record has been approved, data from it is forwarded to the Tivoli Information Management for z/OS TEC Event Adapter running in job BLGTECAD. BLGTECAD invokes TSX BLGTAGSD which generates a TEC event and sents it to TEC via TCP/IP. TEC then runs the PERL script to perform the actual software distribution. The PERL script is described in “PERL Program To Initiate Distribution of a File Package” on page 400.

32

# Tivoli Decision Support

Tivoli Decision Support enables users to determine how effective they are at controlling and managing their problem and change processes. It also enables users to analyze information stored in a Tivoli Information Management for z/OS host database through the Tivoli Decision Support Discovery Interface.

## Where To Find Additional Information

For an overview of how Tivoli Decision Support is used with Tivoli Information Management for z/OS data, refer to the *Tivoli Information Management for z/OS Data Reporting User's Guide*. Installation and usage instructions for Tivoli Decision Support for Information Management are provided in softcopy format on the Tivoli Decision Support for Information Management CD-ROM.

# VI — Appendixes

A

# Relating Publications to Specific Tasks

Your data processing organization can have many different users performing many different tasks. The books in the Tivoli Information Management for z/OS library contain task-oriented scenarios to teach users how to perform the duties specific to their jobs.

The following table describes the typical tasks in a data processing organization and identifies the Tivoli Information Management for z/OS publication that supports those tasks. See "The Tivoli Information Management for z/OS Library" on page 419 for more information about each book.

## Typical Tasks

Table 49. Relating Publications to Specific Tasks

| If You Are: | And You Do This: | Read This: |
|---|---|---|
| Planning to Use Tivoli Information Management for z/OS | Identify the hardware and software requirements of Tivoli Information Management for z/OS. Identify the prerequisite and corequisite products. Plan and implement a test system. | *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* |
| Installing Tivoli Information Management for z/OS | Install Tivoli Information Management for z/OS. Define and initialize data sets. Create session-parameters members. | *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*<br><br>*Tivoli Information Management for z/OS Integration Facility Guide* |
| | Define and create multiple Tivoli Information Management for z/OS BLX-SPs. | *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* |
| | Define and create APPC transaction programs for clients. | *Tivoli Information Management for z/OS Client Installation and User's Guide* |
| | \| Define coupling facility structures for sysplex data sharing. | *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* |
| Diagnosing problems | Diagnose problems encountered while using Tivoli Information Management for z/OS | *Tivoli Information Management for z/OS Diagnosis Guide* |

*Table 49. Relating Publications to Specific Tasks (continued)*

| If You Are: | And You Do This: | Read This: |
|---|---|---|
| Administering Tivoli Information Management for z/OS | Manage user profiles and passwords. Define and maintain privilege class records. Define and maintain rules records. | *Tivoli Information Management for z/OS Program Administration Guide and Reference*<br><br>*Tivoli Information Management for z/OS Integration Facility Guide* |
| | Define and maintain USERS record. Define and maintain ALIAS record. Implement GUI interface. Define and maintain command aliases and authorizations. | *Tivoli Information Management for z/OS Program Administration Guide and Reference* |
| | Implement and administer Notification Management. Create user-defined line commands. Define logical database partitioning. | *Tivoli Information Management for z/OS Program Administration Guide and Reference* |
| | Create or modify GUI workstation applications that can interact with Tivoli Information Management for z/OS. Install the Tivoli Information Management for z/OS Desktop on user workstations. | *Tivoli Information Management for z/OS Desktop User's Guide* |
| Maintaining Tivoli Information Management for z/OS | Set up access to the data sets. Maintain the databases. Define and maintain privilege class records. | *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*<br><br>*Tivoli Information Management for z/OS Program Administration Guide and Reference* |
| | Define and maintain the BLX-SP. Run the utility programs. | *Tivoli Information Management for z/OS Operation and Maintenance Reference* |
| Programming applications | Use the application program interfaces. | *Tivoli Information Management for z/OS Application Program Interface Guide* |
| | Use the application program interfaces for Tivoli Information Management for z/OS clients. | *Tivoli Information Management for z/OS Client Installation and User's Guide* |
| | Create Web applications using or accessing Tivoli Information Management for z/OS data. | *Tivoli Information Management for z/OS World Wide Web Interface Guide* |

*Table 49. Relating Publications to Specific Tasks (continued)*

| If You Are: | And You Do This: | Read This: |
|---|---|---|
| Customizing Tivoli Information Management for z/OS | Design and implement a Change Management system. Design and implement a Configuration Management system. Design and implement a Problem Management system. | *Tivoli Information Management for z/OS Problem, Change, and Configuration Management* |
| | Design, create, and test terminal simulator panels or terminal simulator EXECs. Customize panels and panel flow. | *Tivoli Information Management for z/OS Terminal Simulator Guide and Reference*<br><br>*Tivoli Information Management for z/OS Panel Modification Facility Guide* |
| | Design, create, and test Tivoli Information Management for z/OS formatted reports. | *Tivoli Information Management for z/OS Data Reporting User's Guide* |
| | Create a bridge between NetView and Tivoli Information Management for z/OS applications. Integrate Tivoli Information Management for z/OS with Tivoli distributed products. | *Tivoli Information Management for z/OS Guide to Integrating with Tivoli Applications* |
| Assisting Users | Create, search, update, and close change, configuration, or problem records. Browse or print Change, Configuration, or Problem Management reports. | *Tivoli Information Management for z/OS Problem, Change, and Configuration Management* |
| | Use the Tivoli Information Management for z/OS Integration Facility. | *Tivoli Information Management for z/OS Integration Facility Guide* |
| Using Tivoli Information Management for z/OS | Learn about the Tivoli Information Management for z/OS panel types, record types, and commands. Change a user profile. | *Tivoli Information Management for z/OS User's Guide* |
| | Learn about Problem, Change, and Configuration Management records. | *Tivoli Information Management for z/OS Problem, Change, and Configuration Management* |
| | Receive and respond to Tivoli Information Management for z/OS messages. | *Tivoli Information Management for z/OS Messages and Codes* |
| | Design and create reports. | *Tivoli Information Management for z/OS Data Reporting User's Guide* |

B

# Tivoli Information Management for z/OS Courses

## Education Offerings

Tivoli Information Management for z/OS classes are available in the United States and in the United Kingdom. For information about classes outside the U.S. and U.K., contact your local IBM representative or visit **http://www.training.ibm.com** on the World Wide Web.

### United States

IBM Education classes can help your users and administrators learn how to get the most out of Tivoli Information Management for z/OS. IBM Education classes are offered in many locations in the United States and at your own company location.

For a current schedule of available classes or to enroll, call 1-800-IBM TEACh (1-800-426-8322). On the World Wide Web, visit:

**http://www.training.ibm.com**

to see the latest course offerings.

### United Kingdom

In Europe, the following public courses are held in IBM's central London education centre at the South Bank at regular intervals. On-site courses can also be arranged.

For course schedules and to enroll, call Enrollments Administration on 0345 581329, or send an e-mail note to:

**contact_educ_uk@vnet.ibm.com**

On the World Wide Web, visit:

**http://www.europe.ibm.com/education-uk**

to see the latest course offerings.

# C

# Where to Find More Information

The Tivoli Information Management for z/OS library is an integral part of Tivoli Information Management for z/OS. The books are written with particular audiences in mind. Each book covers specific tasks.

## The Tivoli Information Management for z/OS Library

The publications shipped automatically with each Tivoli Information Management for z/OS Version 7.1 licensed program are:

- *Tivoli Information Management for z/OS Application Program Interface Guide*
- *Tivoli Information Management for z/OS Client Installation and User's Guide **
- *Tivoli Information Management for z/OS Data Reporting User's Guide **
- *Tivoli Information Management for z/OS Desktop User's Guide*
- *Tivoli Information Management for z/OS Diagnosis Guide **
- *Tivoli Information Management for z/OS Guide to Integrating with Tivoli Applications **
- *Tivoli Information Management for z/OS Integration Facility Guide **
- *Tivoli Information Management for z/OS Licensed Program Specification*
- *Tivoli Information Management for z/OS Master Index, Glossary, and Bibliography*
- *Tivoli Information Management for z/OS Messages and Codes*
- *Tivoli Information Management for z/OS Operation and Maintenance Reference*
- *Tivoli Information Management for z/OS Panel Modification Facility Guide*
- *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*
- *Tivoli Information Management for z/OS Program Administration Guide and Reference*
- *Tivoli Information Management for z/OS Problem, Change, and Configuration Management**
- *Tivoli Information Management for z/OS Reference Summary*
- *Tivoli Information Management for z/OS Terminal Simulator Guide and Reference*
- *Tivoli Information Management for z/OS User's Guide*
- *Tivoli Information Management for z/OS World Wide Web Interface Guide*

| **Note:** Publications marked with an asterisk (*) are shipped in softcopy format only.

Also included is the Product Kit, which includes the complete online library on CD-ROM.

To order a set of publications, specify order number SBOF-7028-00.

Additional copies of these items are available for a fee.

Publications can be requested from your Tivoli or IBM representative or the branch office serving your location. Or, in the U.S., you can call the IBM Publications order line directly by dialing 1-800-879-2755.

The following descriptions summarize all the books in the Tivoli Information Management for z/OS library.

***Tivoli Information Management for z/OS Application Program Interface Guide***, SC31-8737-00, explains how to use the low-level API, the high-level API, and the REXX interface to the high-level API. This book is written for application and system programmers who write applications that use these program interfaces.

***Tivoli Information Management for z/OS Client Installation and User's Guide***, SC31-8738-00, describes and illustrates the setup and use of Tivoli Information Management for z/OS's remote clients. This book shows you how to use Tivoli Information Management for z/OS functions in the AIX, CICS, HP-UX, OS/2®, Sun Solaris, Windows NT, and OS/390 UNIX System Services environments. Also included in this book is complete information about using the Tivoli Information Management for z/OS servers.

***Tivoli Information Management for z/OS Data Reporting User's Guide***, SC31-8739-00, describes various methods available to produce reports using Tivoli Information Management for z/OS data. It describes Tivoli Decision Support for Information Management (a Discovery Guide for Tivoli Decision Support), the Open Database Connectivity (ODBC) Driver for Tivoli Information Management for z/OS, and the Report Format Facility. A description of how to use the Report Format Facility to modify the standard reports provided with Tivoli Information Management for z/OS is provided. The book also illustrates the syntax of report format tables (RFTs) used to define the output from the Tivoli Information Management for z/OS REPORT and PRINT commands. It also includes several examples of modified RFTs.

***Tivoli Information Management for z/OS Desktop User's Guide***, SC31-8740-00, describes how to install and use the sample application provided with the Tivoli Information Management for z/OS Desktop. The Tivoli Information Management for z/OS Desktop is a Java-based graphical user interface for Tivoli Information Management for z/OS. Information on how to set up data model records to support the interface and instructions on using the Desktop Toolkit to develop your own Desktop application are also provided.

***Tivoli Information Management for z/OS Diagnosis Guide***, GC31-8741-00, explains how to identify a problem, analyze its symptoms, and resolve it. This book includes tools and information that are helpful in solving problems you might encounter when you use Tivoli Information Management for z/OS.

***Tivoli Information Management for z/OS Guide to Integrating with Tivoli Applications***, SC31-8744-00, describes the steps to follow to make an automatic connection between NetView and Tivoli Information Management for z/OS applications. It also explains how to customize the application interface which serves as an application enabler for the NetView Bridge and discusses the Tivoli Information Management for z/OS NetView AutoBridge. Information on interfacing Tivoli Information Management for z/OS with other Tivoli management software products or components is provided for Tivoli Enterprise Console, Tivoli Global Enterprise Manager, Tivoli Inventory, Tivoli Problem Management, Tivoli Software Distribution, and Problem Service.

***Tivoli Information Management for z/OS Integration Facility Guide***, SC31-8745-00, explains the concepts and structure of the Integration Facility. The Integration Facility provides a task-oriented interface to Tivoli Information Management for z/OS that makes the

Tivoli Information Management for z/OS applications easier to use. This book also explains how to use the panels and panel flows in your change and problem management system.

***Tivoli Information Management for z/OS Master Index, Glossary, and Bibliography***, SC31-8747-00, combines the indexes from each hardcopy book in the Tivoli Information Management for z/OS library for Version 7.1. Also included is a complete glossary and bibliography for the product.

***Tivoli Information Management for z/OS Messages and Codes***, GC31-8748-00, contains the messages and completion codes issued by the various Tivoli Information Management for z/OS applications. Each entry includes an explanation of the message or code and recommends actions for users and system programmers.

***Tivoli Information Management for z/OS Operation and Maintenance Reference***, SC31-8749-00, describes and illustrates the BLX-SP commands for use by the operator. It describes the utilities for defining and maintaining data sets required for using the Tivoli Information Management for z/OS licensed program, Version 7.1.

***Tivoli Information Management for z/OS Panel Modification Facility Guide***, SC31-8750-00, gives detailed instructions for creating and modifying Tivoli Information Management for z/OS panels. It provides detailed checklists for the common panel modification tasks, and it provides reference information useful to those who design and modify panels.

***Tivoli Information Management for z/OS Planning and Installation Guide and Reference***, GC31-8751-00, describes the tasks required for installing Tivoli Information Management for z/OS. This book provides an overview of the functions and optional features of Tivoli Information Management for z/OS to help you plan for installation. It also describes the tasks necessary to install, migrate, tailor, and start Tivoli Information Management for z/OS.

***Tivoli Information Management for z/OS Problem, Change, and Configuration Management***, SC31-8752-00, helps you learn how to use Problem, Change, and Configuration Management through a series of training exercises. After you finish the exercises in this book, you should be ready to use other books in the library that apply more directly to the programs you use and the tasks you perform every day.

***Tivoli Information Management for z/OS Program Administration Guide and Reference***, SC31-8753-00, provides detailed information about Tivoli Information Management for z/OS program administration tasks, such as defining user profiles and privilege classes and enabling the GUI user interface.

***Tivoli Information Management for z/OS Reference Summary***, SC31-8754-00, is a reference booklet containing Tivoli Information Management for z/OS commands, a list of p-words and s-words, summary information for PMF, and other information you need when you use Tivoli Information Management for z/OS.

***Tivoli Information Management for z/OS Terminal Simulator Guide and Reference***, SC31-8755-00, explains how to use terminal simulator panels (TSPs) and EXECs (TSXs) that let you simulate an entire interactive session with a Tivoli Information Management for z/OS program. This book gives instructions for designing, building, and testing TSPs and TSXs, followed by information on the different ways you can use TSPs and TSXs.

***Tivoli Information Management for z/OS User's Guide***, SC31-8756-00, provides a general introduction to Tivoli Information Management for z/OS and databases. This book has a series of step-by-step exercises to show beginning users how to copy, update, print, create, and delete records, and how to search a database. It also contains Tivoli Information Management for z/OS command syntax and descriptions and other reference information.

***Tivoli Information Management for z/OS World Wide Web Interface Guide***, SC31-8757-00, explains how to install and operate the features available with Tivoli Information Management for z/OS that enable you to access a Tivoli Information Management for z/OS database using a Web browser as a client.

Other related publications include the following:

***Tivoli Decision Support: Using the Information Management Guide*** is an online book (in portable document format) that can be viewed with the Adobe Acrobat Reader. This book is provided with Tivoli Decision Support for Information Management (5697-IMG), which is a product that enables you to use Tivoli Information Management for z/OS data with Tivoli Decision Support. This book describes the views and reports provided with the Information Management Guide.

IBM Redbooks™ published by IBM's International Technical Support Organization are also available. For a list of redbooks related to Tivoli Information Management for z/OS and access to online redbooks, visit Web site **http://www.redbooks.ibm.com** or **http://www.support.tivoli.com**

# Index

File Number: S370/30xx/4300
Program Number: 5697-SD9

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC31-8744-00