Information Management for z/OS

Application Program Interface Guide

Version 7.1

SC31-8737-00

Information Management for z/OS

Application Program Interface Guide

Version 7.1

SC31-8737-00

Tivoli Information Management for z/OS Application Program Interface Guide

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

The Tivoli® Information Management for z/OS Application Program Interfaces (APIs) provide a means of providing data through defined data structures that you can create and process with your application programs. This guide describes APIs and tells how your applications can use them to access the database.

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

If you are a system or application programmer, you can use this publication as a guide and reference in writing application programs that access and run Tivoli Information Management for z/OS database functions.

You should be familiar with the information in the *Tivoli Information Management for z/OS User's Guide* and the *Tivoli Information Management for z/OS Program Administration Guide and Reference* before you use this guide. You should also be familiar with the basics of Tivoli Information Management for z/OS's problem management, change management, and configuration management facilities. If you are developing and modifying Desktop or Web applications, you should also be familiar with the concepts of API return and reason codes. "Where to Find More Information" on page 377 lists the publications that contain information about these subjects.

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 377.

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

This guide is structured as follows:

- "Introduction to the Application Program Interfaces" on page 1 describes the Low- and High-Level Application Program Interfaces (LLAPI, HLAPI), in terms of their functions, components, and operating characteristics.
- "Using the LLAPI" on page 15 explains the transactions that the low-level interface uses. The chapter also provides step-by-step instructions for creating transactions to perform typical tasks, describes interface structures and interface tables, and it describes the relationships among components of the interface.
- "Using the HLAPI" on page 145 explains the transactions that the high-level interface uses, and how to use the HLAPI/REXX interface. The chapter also provides step-by-step instructions for creating transactions to perform typical tasks, describes interface structures and interface tables, and it describes the relationships among components of the interface.
- "HLAPI Extensions" on page 263 describes a specific extension to the HLAPI which may cause fewer transactions to the server, thereby improving performance. This chapter also describes information about how to write other HLAPI TSX extensions.
- "Tips for Writing an API Application" on page 273 describes the steps typically involved in creating an application that uses the Tivoli Information Management for z/OS APIs.
- "Field Validation Using the Field Validation Module BLGPPFVM" on page 279 describes the Table Build Utility (BLGUT8) you use to build data, pattern, and alias

tables for use by the interfaces. This chapter also explains how to validate data fields using the Field Validation Module (BLGPPFVM).

- "API Control Flow" on page 283 describes some of the logic of the APIs. This chapter also discusses the two modes of operation of the LLAPI, panel processing and bypass panel processing.
- "API Security" on page 287 describes the security checking available to ensure that a user has the authority to use the value specified in **PICAUSRN**.
- "Tailoring the Application Program Interfaces" on page 289 describes how you can modify the interfaces to better meet the needs of your installation.
- "LLAPI User Exits" on page 293 describes the user exits used in the API environment.
- The appendixes in this guide give you reference information that you might find useful when you are working with the APIs. You can find return and reason codes, transaction lists, and more.

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 guide, selections on selection or options panels and fields on data-entry panels appear **like this**. The input you enter in response to the fields on data-entry panels appears like this.

Commands, such as END, CONTROL, RESUME, or FIELD, appear as illustrated. Although not commands, the user responses YES and NO also appear in the same highlighting as commands.

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**.

1

Introduction to the Application Program Interfaces

Tivoli Information Management for z/OS extends your ability to gather, organize, and locate information about your company's data processing installation. The Tivoli Information Management for z/OS application program interfaces (APIs) are part of this product.

The main purpose of the Tivoli Information Management for z/OS database is to hold problem, change, configuration, and user-defined data for your company. Traditionally, Tivoli Information Management for z/OS data has been managed through panel interfaces, either with direct user interaction or with Terminal Simulator Panels (TSPs) acting for users.

Of course, you can still manage your database that way, but sometimes you cannot or do not want to access your information interactively. Sometimes you might want to use applications that are external to Tivoli Information Management for z/OS to access this database.

The Tivoli Information Management for z/OS APIs accept and provide data through defined data structures that you can create and process with your application programs. The APIs enable you to use external applications to perform Tivoli Information Management for z/OS record access functions. You can extract or enter data into the Tivoli Information Management for z/OS database from external sources. The APIs enable you to have an ongoing dialog between an external resource and your Tivoli Information Management for z/OS database. With the APIs, it is possible for you to open up the Tivoli Information Management for z/OS environment and transfer data between your external sources and the Tivoli Information Management for z/OS database.

Compared to earlier, batch-oriented methods, using the APIs provides you control advantages over your data operations. The APIs enable real-time processing that is synchronous with your application. You gain better control because you can check your data streams as well as determine if your operations were successful.

Using the APIs does not require an interactive user, so the Tivoli Information Management for z/OS requirements for Time Sharing Option (TSO) and Interactive System Productivity Facility (ISPF) do not apply to API uses. Most of the time, a storage region size of 4MB (1MB equals 1 048 576 bytes) is sufficient for API use. If you have installed the Tivoli Information Management for z/OS NetView® Bridge Adapter, you might need to increase the region size to 6MB.

Tivoli Information Management for z/OS supports two APIs. They are the Low-Level Application Program Interface (LLAPI) and the High-Level Application Program Interface (HLAPI). Both APIs perform the same tasks. However, for a given task, the HLAPI requires fewer user-written instructions. The LLAPI requires more user-written instructions, but it

provides a greater degree of interface control. The HLAPI has an additional facility called the HLAPI/REXX interface. The HLAPI/REXX interface enables you to access HLAPI functions from REXX code. Using these APIs, your external applications can access and manipulate Tivoli Information Management for z/OS's problem, change, and configuration database records, as well as your own user-defined records.

Tivoli Information Management for z/OS supports access from applications running on other operating systems through the HLAPI. Three MVS™-based servers are provided to support access from remote clients:

- The Remote Environment Server (RES), which serves one client at a time, and uses advanced program-to-program communication (APPC).
- The Multiclient Remote Environment Server with APPC (MRES with APPC), which can serve multiple clients concurrently, and uses advanced program-to-program communication.
- The Multiclient Remote Environment Server with TCP/IP (MRES with TCP/IP), which can serve multiple clients concurrently, and uses TCP/IP.

These servers provide access to Tivoli Information Management for z/OS data for client application programs running on Operating System/2® (OS/2®), CICS®, UNIX®, and Windows NT® platforms. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for more information on the servers and clients, including information on the communication protocols the clients support.

Writing Applications for the APIs

If you write applications that process customized Tivoli Information Management for z/OS records, you might need to perform setup steps or tailor the API TSPs to correctly process these records. See "Tips for Writing an API Application" on page 273, "Tailoring the Application Program Interfaces" on page 289, and "Terminal Simulator Panels" on page 349 for more information.

Any application you write that uses the APIs must perform three basic steps: API initialization, API processing, and API termination. This sequence of steps is called a *session*. The individual interactions and data accesses of each session are called *transactions*. The API initialization and API termination steps are individual transactions. They are started only once for each session. The HLAPI/REXX interface initializes and terminates a session for you when you use it. However, your application must still perform the initialize and terminate transactions. The API processing step can consist of many transactions. This step makes up the bulk of any session. The following sections discuss these basic steps, writing applications for remote environments, and security.

Initializing

Initialization involves specifying the characteristics of this API session and establishing the Tivoli Information Management for z/OS environment.

Your application loads a server module that is supplied by Tivoli Information Management for z/OS into the application's address space. The application calls the module with a defined set of initialization parameters.

Note: For performance reasons, you can load the module once, before the API initialization transaction, call it as many times as you need it, then delete it after the API termination transaction. This saves any processing overhead that is required for reading the module from the disk many times.

When using the HLAPI/REXX interface, your REXX program links to the HLAPI/REXX interface server module. It, in turn, accesses the HLAPI server module.

The server then loads and initializes the remainder of the code (the subtask) necessary to access the Tivoli Information Management for z/OS database. The server and the supporting code for accessing the Tivoli Information Management for z/OS database run as extensions of the application program and use resources in the application's address space. For these reasons, your application, the API, and the database must reside on the same system.

Note: Tivoli Information Management for z/OS uses system services such as GETMAIN to acquire resources and therefore might not be appropriate for running under certain subsystems.

To initialize the Tivoli Information Management for z/OS environment, you must use the transaction code to initialize Tivoli Information Management for z/OS (transaction T001 for the LLAPI, HL01 for the HLAPI, or INIT for the HLAPI/REXX interface). This prepares Tivoli Information Management for z/OS for further transaction processing.

Processing

The processing portion of your application involves several steps. First, you specify the action that you want to perform and any data or options needed to perform that action. Then you perform the action. Finally, you process any data that returns to your application.

You can use applications that start the APIs to retrieve, create, update, inquire about, and delete records in the Tivoli Information Management for z/OS database. You can write your application programs in languages such as C, PL/I, COBOL, C++, assembler, or any program language that supports the data structures of the APIs. If you are using a remote platform, you can also write your application program in Java™. See "LLAPI Structures" on page 100 and "HLAPI Structures" on page 216 for information about these structures. See the *Tivoli Information Management for z/OS Client Installation and User's Guide* and the *Tivoli Information Management for z/OS World Wide Web Interface Guide* for additional information about accessing the Tivoli Information Management for z/OS database from a remote platform. You can also use the REXX language to communicate with the HLAPI through the HLAPI/REXX interface. To perform the API functions, call again the server that you used to initialize the API and give it the appropriate transaction code to perform the function you want. The API remains initialized and ready to use for any number and any combination of processing transactions as long as your application remains active, or until you perform a termination transaction.

Terminating

Termination of the Tivoli Information Management for z/OS environment means that you close down your API session and the Tivoli Information Management for z/OS environment. Your application closes the environment by issuing the termination transaction. This transaction returns to the system all resources that the server acquired and deletes the associated database access code from the address space. You must run the termination transaction under the same task control block that the initialization transaction was run under.

If you are using MVS and you stop the API, it is not available for use until you initialize it again. Repeated initializations and terminations can affect performance, so perform as few of them as possible.

CICS Applications

You can develop CICS applications that issue Tivoli Information Management for z/OS HLAPI calls for any Tivoli Information Management for z/OS database. You can do this by using the Tivoli Information Management for z/OS High-Level Application Program Interface Client for CICS (HLAPI/CICS), a remote environment client that Tivoli Information Management for z/OS supports. This client does not extend the CICS function set; it enables CICS transactions to retrieve and update Tivoli Information Management for z/OS data. Of the three MVS-based servers described on page 2, the HLAPI/CICS client can connect to either the Remote Environment Server (RES) or the Multiclient Remote Environment Server with APPC (MRES with APPC). Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for more information on the Tivoli Information Management for z/OS servers and installing and using the HLAPI/CICS client.

OS/2 Applications

You can develop OS/2 applications that issue Tivoli Information Management for z/OS HLAPI calls for any Tivoli Information Management for z/OS database. You can do this by using the HLAPI/2, a remote environment client that Tivoli Information Management for z/OS supports. This client does not extend the OS/2 function set; it enables OS/2 applications to retrieve and update Tivoli Information Management for z/OS data. The HLAPI/2 client connects to any of the Tivoli Information Management for z/OS servers listed on page 2. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for more information on the Tivoli Information Management for z/OS servers and installing and using the HLAPI/2 client.

The OS/2 remote environment client also provides a REXX interface that permits you to access HLAPI/2 from REXX programs. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for more information about this interface.

UNIX Applications

You can develop UNIX applications that issue Tivoli Information Management for z/OS HLAPI calls for any Tivoli Information Management for z/OS database. You can do this by using one of the remote environment clients on AIX®, HP-UX, and Sun Solaris that Tivoli Information Management for z/OS supports or by using the client . These clients do not extend the UNIX function sets; they enable UNIX applications to retrieve and update Tivoli Information Management for z/OS data. The HLAPI/AIX client connects to any of the Tivoli Information Management for z/OS servers listed on page 2. The HLAPI/HP, HLAPI/Solaris, and HLAPI for OS/390 UNIX System Services (HLAPI/USS) connect to only the MRES with TCP/IP. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for more information on the Tivoli Information Management for z/OS servers and installing and using the HLAPI/UNIX clients.

The AIX remote environment client (HLAPI/AIX) provides a REXX interface that allows you to access HLAPI/AIX from REXX programs. The HLAPI/USS also provides a REXX interface that allows you to access HLAPI/USS from REXX programs. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for more information on this interface.

Windows NT Applications

You can develop Windows NT applications that issue Tivoli Information Management for z/OS HLAPI calls for any Tivoli Information Management for z/OS database. You can do this by using the HLAPI/NT, a remote environment client that Tivoli Information

Management for z/OS supports. The HLAPI/NT connects to any of the Tivoli Information Management for z/OS servers listed on page 2. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for more information on the Tivoli Information Management for z/OS servers and installing and using the HLAPI/NT client.

Java Applications

You can run Java application programs that interface with the Tivoli Information Management for z/OS Client Application Programming Interfaces (CAPIs) on any operating system platform that supports both the Tivoli Information Management for z/OS Client APIs and the Java Version 1.1.1 run-time environment. A sample Java program that illustrates the use of the Java class objects is also provided with the clients.

The *Tivoli Information Management for z/OS Client Installation and User's Guide* contains additional information on Java classes and methods that are distributed with Tivoli Information Management for z/OS.

Security

For additional security over the APIs, keep the server modules (BLGYSRVR and BLGYHLPI) in a limited access library using security software, such as RACF®. The module that provides access to the HLAPI from REXX programs is BLGYRXM. Place it, too, in a limited access library in the same manner. In this way you can prevent unauthorized applications from accessing the Tivoli Information Management for z/OS database through the APIs. Additional information regarding security aspects of the API can be found in "API Security" on page 287.

Date Format

Your APIs can use any supported date format and have dates converted to or from the format used in the database. For the LLAPI, this is controlled by the PICADFMT flag in the PICA (where the PICADSEP must also be specified). For the HLAPI, this is controlled by the use of a PDB named DATE_FORMAT. The date formats available for the LLAPI are described in the discussion of the PICA flag PICADFMT, described on page 111; the date formats available for the HLAPI are discussed throughout "Using the HLAPI" on page 145.

The Low-Level Application Program Interface

The LLAPI runs in the MVS environment and consists of the following components:

- Server (callable enabler)
- Structures and tables
- Tivoli Information Management for z/OS API subtask code

Understanding the LLAPI Control and Data Flow

Your application must establish linkage with the server, allocate and prepare the communications area, and interact with the server and LLAPI tables. The server validates transaction data and attaches to and interfaces with the Tivoli Information Management for z/OS API subtask. The subtask is attached to the server automatically when the server is initialized. The API subtask performs transaction processing, table construction and navigation, data set accessing, and error processing.

Figure 1 on page 6 shows the components of the LLAPI and the interface points between them. In addition, the figure shows the key elements within the structures and tables component. The structures and tables are described in detail in "LLAPI Structures" on page 100.

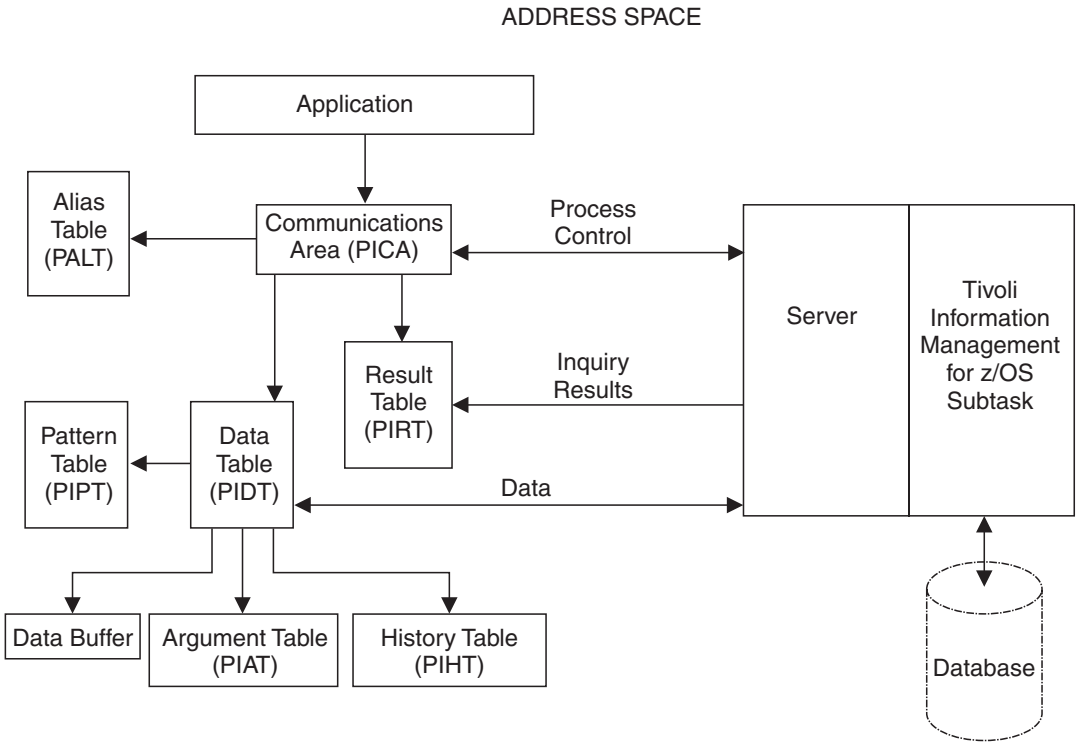

Figure 1. LLAPI Components and Data Flows

All communications between your application and the LLAPI take place through transactions. Each transaction uses specific control blocks and structures to convey information between the application and the interface. This forms the basis for control flow. The transaction code tells the LLAPI what to do, and the call to the server tells the LLAPI when to do it. Because your application initiates all calls to the LLAPI, it follows that the application dictates the control flow.

PICA

Data flow is synchronized when the transaction runs. Therefore, it too is controlled by your application. All transactions use a set of tables and data structures. The first and most significant of these structures is the program interface communications area (PICA). In Figure 1 it is labeled Communications Area. Your application and the LLAPI both use the PICA as the focal point for data specification.

The PICA is the only LLAPI control block for which your application must allocate storage. It is where you specify your initialization parameters and transaction codes. It is where you receive your return and reason codes. And it also serves as the anchor to all other LLAPI structures. When you enable the interface by calling the server to start a transaction, the only parameter you pass in the call is the address of the parameter list that points to the PICA. All other information necessary to complete the transaction is either contained in, or located through, the PICA. See “Low-Level Program Interface Communications Area (PICA)” on page 101 for a detailed description of the communications area.

The LLAPI uses other structures and data to support the transactions that your applications use to access the Tivoli Information Management for z/OS database. These structures and a core set of data are called *resources*. With the exception of the PICA, the LLAPI acquires

these resources through transactions initiated by your application. Not all of the tables shown in Figure 1 on page 6 are used for every transaction. The tables used by the LLAPI depend on which transaction your application wants to perform.

PIDT

The PICA identifies the program interface data table (PIDT) for the LLAPI and your application. The PIDT is labeled Data Table in Figure 1 on page 6. The PIDT defines a view of a Tivoli Information Management for z/OS database record. Based on the data contained in the record, you can provide a pre-defined (or *static*) view, request that the LLAPI build a *dynamic* view, or direct the LLAPI to generate a PIDT from data model records. For detailed information about the PIDT, see "Program Interface Data Table (PIDT)" on page 114. For more information about types of PIDTs and BLGUT8, see "Field Validation Using the Field Validation Module BLGPPFVM" on page 279.

PIPT

The PIDT also identifies the characteristics of the fields and serves as the anchor for the program interface pattern table (PIPT), labeled Pattern Table in Figure 1 on page 6. Except for dynamic PIDTs, the PIPT contains the validation criteria for the data associated with the indexes and the data buffer. The data buffer contains the data itself and is anchored to the PIDT.

PIAT

For inquiry transactions, the PIDT identifies the program interface argument table (PIAT), labeled Argument Table in Figure 1 on page 6. You can use the Argument Table to specify freeform search arguments.

PIHT

For retrieve transactions that request processing of history data, the PIDT identifies the program interface history table (PIHT), labeled History Table in Figure 1 on page 6. You can use the History Table to modify or input history data on subsequent update and create record transactions. For detailed information about the PIHT, see "Program Interface History Table (PIHT)" on page 132.

PALT

The program interface alias table (PALT), labeled Alias Table in Figure 1 on page 6, is identified through the PICA. The Alias Table lets your application specify alias names for PIDTs, p-words, p-word indexes, and s-word indexes. It also enables you to specify default values for fields. "Alias Tables" on page 238 gives you a description of how the HLAPI uses this table. That description can serve as an example of how you can use the table in an application for the LLAPI. For more detailed information about the PALT see "Program Interface Alias Table (PALT)" on page 112.

PIRT

Inquiry transactions require the use of the program interface results table (PIRT), labeled Result Table in Figure 1 on page 6. It contains the record IDs of records that were found to meet an application's specific search criteria. The PIRT, is also identified through the PICA. For detailed information about the PIRT, see "Program Interface Results Table (PIRT)" on page 141.

A Typical Scenario

Suppose you are writing an application to retrieve and display database problem records that meet specific criteria. You establish criteria for problem records that are:

- Nonclosed

- Associated with a particular department
- Assigned dates within a specified range

Note: This example is specifically designed, for instructional purposes, to retrieve and display records. If all you want to do is find a record ID and any one other piece of data in a record (maximum length 45 characters), the search can accomplish this without any retrieves at all.

To do this, ensure that your application performs the following steps:

1. Specify the characteristics of this API session and establish the Tivoli Information Management for z/OS environment.
2. Request problem record inquiry resources from the interface so that the application can define to the interface the specific criteria it is looking for.

   Your application can specify the following two forms of inquiry, or a combination of both forms:
   - A simulation of the interactive quick-search response entry
   - A simulation of the interactive freeform argument response entry

   For more information on inquiry argument generation, see "Record Inquiry (T107)" on page 84.
3. After defining the specific inquiry criteria by one of the methods in the previous step, your application passes the inquiry specifications to the API by using an inquiry transaction.
4. The API searches the database and passes a list of the external problem record IDs that match the specific inquiry criteria back to your application in the Results Table, or PIRT.
5. Your application then requests the API to retrieve each record that was in the list. As each retrieve transaction is performed, the API converts the record data to an external form that uses a data buffer and data table (PIDT) structure. The PICA provides the information to locate the PIDT, and the PIDT provides the information on how to locate the data in the Data Buffer. Your application can use whatever transactions are necessary to display the records while still in this API session.
6. After your records have been displayed, your application can continue to perform other tasks for you, or it can end this session.

You can find a complete example of a program that uses the LLAPI in the Tivoli Information Management for z/OS SAMPLIB (MVS data set SBLMSAMP). Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for information on the person to contact for information on high-level qualifiers of data sets at your site.

The High-Level Application Program Interface

The HLAPIHLAPI consists of the following components:

- Server (callable enabler)
- High-Level communication area structure (HICA)
- Parameter data blocks (PDBs)

While not actually parts of the HLAPI, the LLAPI and API subtask components are used by the HLAPI to perform its functions.

You can use the HLAPI from MVS, or you can use it from user application programs running on a different operating system in a remote environment. User application programs interact with Tivoli Information Management for z/OS from a remote environment in basically the same way as they do from MVS using the HLAPI. Additional information on these client applications can be found in the *Tivoli Information Management for z/OS Client Installation and User's Guide*.

Understanding the HLAPI Control and Data Flow

The HLAPI enables you to use a set of callable high-level service functions that the interface transforms into LLAPI transactions that it passes to the LLAPI. Using the HLAPI reduces the complexity of your applications because a single HLAPI transaction can cause several LLAPI transactions to run.

Figure 2 shows, for an MVS operating system, where the HLAPI fits into the common address space occupied by the APIs, your MVS application, and the server/subtask. Figure 2 also illustrates their relationships and how the data flows between them. From HLAPI transactions started by your application, the HLAPI generates LLAPI transactions that are processed by the LLAPI and the subtask. The subtask and LLAPI then return processing results and other information, such as messages and error codes, back through the HLAPI to your application. You cannot mix HLAPI transactions and LLAPI transactions during the same session.

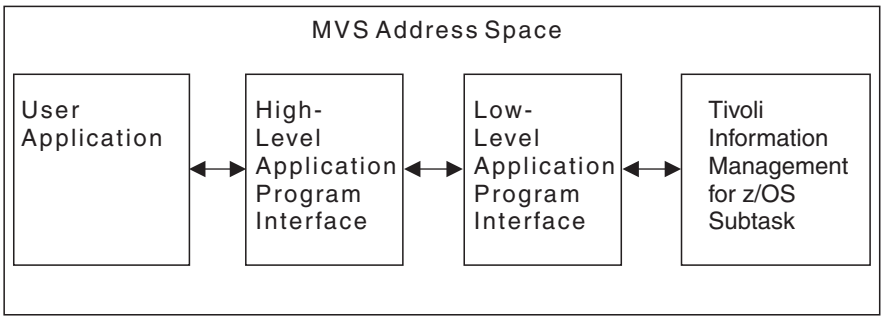

Figure 2. HLAPI Relationships to Your Application and the LLAPI

Figure 3 on page 10 shows the components of the HLAPI and the data flow relationships between them.

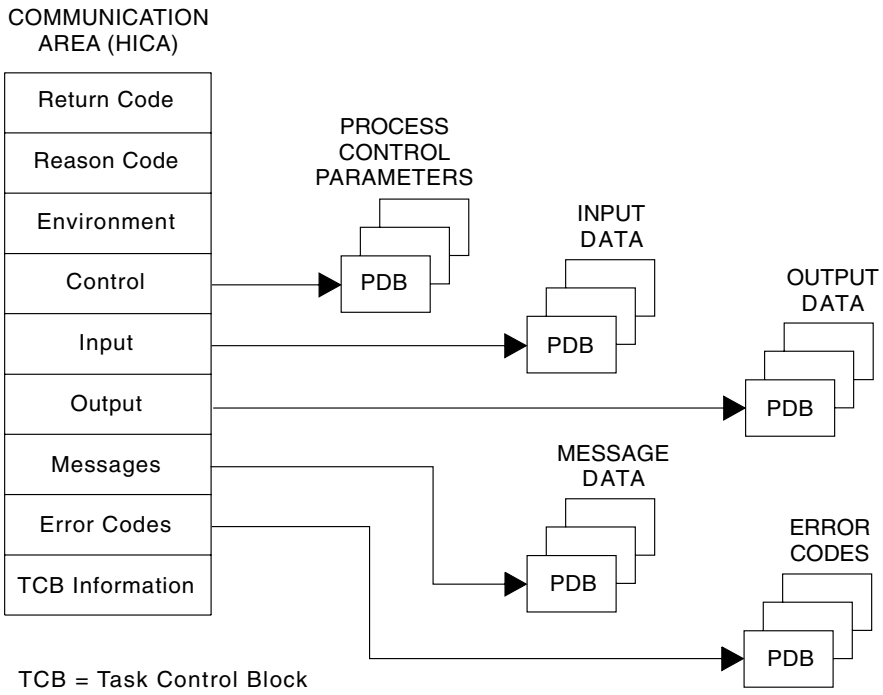

Figure 3. HLAPI Components

The main components of the HLAPI are the HICA structure and parameter data blocks (PDBs) for control, input, output, message, and error chains. The HICA serves the same purpose for the HLAPI that the program interface communications area (PICA) serves for the LLAPI. The HICA:

- Provides a place where you specify your initialization parameters and transaction codes
- Receives return and reason codes
- Anchors the PDBs

The PDBs provide a common structure for communication between your application and the HLAPIHLAPI, replacing the various tables and buffers that the LLAPI uses. The HLAPI uses this structure for both control information and data, as well as for input and output communications. Your application must allocate and initialize a PDB for each item of data that it passes to the HLAPI. The API allocates and initializes a PDB for each data item that it passes back. Every time your application calls the HLAPI, the API frees the PDBs that were used in the transaction prior to the latest call, so your application must completely process the PDB chains it receives before it requests a new HLAPI transaction. These components are described in detail in “HLAPI Structures” on page 216.

You can find a complete example of a program that uses the HLAPI in the Tivoli Information Management for z/OS SAMPLIB (MVS data set SBLMSAMP). Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for information on the person to contact for information on high-level qualifiers of data sets at your site. See “Sample HLAPI/REXX Interface” on page 369 and “Sample High-Level Application Program Interface” on page 367 for more information on sample programs.

Choosing the Appropriate API

Because the LLAPI and the HLAPI perform the same functions, choosing the one to use must be based on API characteristics other than function. Table 1 lists characteristics of both the LLAPI and the HLAPI. Comparing these characteristics might help you to choose the API most appropriate to your needs.

Table 1. Characteristics of the LLAPI and HLAPI

LLAPI Characteristics	HLAPI Characteristics
Offers more user control.	Offers less user control.
Only available from MVS	Available from remote platforms. See the *Tivoli Information Management for z/OS Client Installation and User's Guide* for more information on the use of remote platforms for accessing the Tivoli Information Management for z/OS database.
Enables synchronous or asynchronous operation.	Enables synchronous operation for HLAPI on MVS and asynchronous operation for HLAPI on some remote platforms.
Enables alias processing. (Alias processing must be performed by your application.)	Enables automatic alias processing.
You need more transactions to do many functions (tasks).	You need fewer transactions to do many functions (tasks) because many HLAPI transactions are converted into multiple LLAPI transactions.
There are more control blocks and structures to understand.	There are fewer control blocks and structures to understand.
Uses less storage.	Uses more storage because of PDB allocation.
Enables specific field retrieval only by using customized PIDTs. Your application code must scan through the PIDT to select fields if you do not use a customized PIDT.	Enables specific field retrieval by using a field retrieval list. The HLAPI can scan through the PIDT and return selected fields in PDBs.
Enables data validation by allowing you to call Field Validation Module BLGPPFVM for each field to be validated.	Enables automatic data validation through automatic calls to the BLGPPFVM module.
Does not enable use of REXX programming language to access functions.	Enables use of REXX programming language to access functions.
Session does not end when a transaction timeout condition occurs.	Session ends when a transaction timeout condition occurs.
Enables dynamic PIDT processing.	Does not enable dynamic PIDT processing.
Can process history data.	Can only retrieve history data and delete data based on date.

Data Model Records

The *composition* of data records is that set of fields which define a record type and the attributes of each of those fields. Tivoli Information Management for z/OS utilizes two means of describing the composition of data records: panels and data model records.

When *panels* are used, there is a limitation because the API cannot access the composition directly; the composition must be extracted by the utility BLGUT8 and stored in static tables (PIDTs and PIPTs). These static tables, the PIDTs and PIPTs, convey the data view and field attributes of data to be used by applications which used the APIs.

Data model records provide a means of storing the composition of data records in records rather than in panels.

There are three types of data model records:

Data view records

Data view records describe the entire record's contents. Information needed for certain record transactions, such as the record-type s-word and authorization codes, are included in these records. Data view records contain data attribute record IDs. Data view records replace static PIDTs and PIPTs and are used to generate the in-storage PIDTs and PIPTs. These records are only used by the APIs.

To specify a data view record in the LLAPI, the name is passed in the existing PIDT name field PICATABN and a flag, PICADMRC=Y, is set to indicate that a PIDT should be built using the data view record and given the data view record ID as its name. If you are using the HLAPI, specify the data view record ID in the control PDB DATA_VIEW_NAME.

It can take a significant amount of time to generate a PIDT from data view records. The length of time depends on the number of data attribute records (and validation record that they reference) contained in the data view record. Therefore, it can be especially important to direct the HLAPI to maintain PIDTs in storage if you are using data model records.

Data attribute records

Data attribute records describe the data and its attributes; a data attribute record takes the place of an assisted-entry panel. There are several methods of using data attribute records on an interactive panel sequence. Refer to the *Tivoli Information Management for z/OS Panel Modification Facility Guide* for more information on using data attribute records.

Note: If data attribute records are used as direct-add fields, then normal file processing is not performed for change records when change approval processing is being performed. That is, if ALL of these five direct-adds—DATE/, TIME/, CLAE/, DATM/, and TIMM/—are changed to data attribute records, then date modified, time modified, and user ID are not saved in the record

Data attribute records can be used with data view records for use with the API.

Validation records

Validation records are records that contain validation criteria for a field in a record.

These records can be used with the APIs. In addition, validation records include validation patterns, prefixes, authorization codes, and group prefix indicators to contain all of the data necessary for data validation. The utility BLGUT8 copies the validation record ID, validation s-word, and validation data s-word into a static PIDT. All of this information is put into the PIDT generated from a specific data view record. When the PIDT is loaded or generated for use with a transaction within an API, the validation data is resolved, except for the validation record ID. Because

the contents of the record must be available to find the validation record ID when one is specified, the validation record ID is resolved when processing the PIDT entry. You should place the entry with the validation record ID s-word following the entry which contains the s-word that is used to find the validation record ID. For example, if field “system name” has a validation record ID s-word that is the “location” field, the system name entry must follow the location entry in the static PIDT or data view record so that the LLAPI can find the location record ID to use to validate the system name data.

If you use data model records, the following program exits can be invoked when data is entered into fields. There is a limitation that only one program exit can be invoked for a field.

- BLG01052
- BLG01054
- BLG01147
- BLG01246
- BLG01273
- BLG01437
- BLG01438
- BLG01439
- BLG02024
- BLG02096
- BLG02097
- BLG02119
- BLG02120
- BLG02121

Refer to the *Tivoli Information Management for z/OS Panel Modification Facility Guide* for more information on using data model records.

2

Using the LLAPI

This chapter tells you how your applications can use the LLAPI to access a Tivoli Information Management for z/OS database to perform the tasks of:

- Creating records
- Updating records
- Retrieving records
- Inquiring about records
- Deleting records
- Using TSPs and TSXs

Although your application must interface with external systems that provide the data and other parameters on which you base your use of the LLAPI, this aspect of your application is not discussed. Instead, this book assumes that such data and parameters are present to cause your application to start LLAPI transactions to accomplish the tasks in the preceding list. In addition, this chapter does not discuss how your application processes transaction results and what your application does next.

LLAPI text data set attributes are modified. A description of the changes follows in “Data Sets” on page 21.

LLAPI Operating Characteristics

Using the LLAPI entails certain operating restrictions and characteristics.

Control Transfer Considerations

In an MVS environment, most high-level languages create an internal parameter list structure in which the first 4 bytes are the address of the PICA. The call to the server passes the PICA structure itself and not its address. See Figure 4 on page 21 for more information.

Also consider how to transfer control to the LLAPI server (module BLGYSRVR). BLGYSRVR is installed with attributes of AMODE 31, RMODE ANY. Starting with Version 5.1, Tivoli Information Management for z/OS runs above the 16MB address range. Applications must be either link-edited with AMODE(31) RMODE(ANY) or modified to use the MVS LINK macro to transfer control to BLGYSRVR. Also consider establishing the server entry address by preloading BLGYSRVR using either the MVS LOAD macro or the equivalent function in the language that you are writing your application. This method is usually most efficient because the server is loaded into storage only once, thereby saving load I/O cycles. Consider, too, whether you want to enable the LLAPI to return data above the 16MB address range.

Operating Modes

The LLAPI works in synchronous or asynchronous modes. In synchronous mode, your application does not receive control until the LLAPI returns transaction status

to your application. In asynchronous mode, the LLAPI returns control to your application as soon as it receives a transaction request. Using asynchronous mode enables your application to perform other tasks while the LLAPI is operating. Your application can check on the status of the requested transaction at a later point in the application program flow by using the check transaction completion (T010) function. Except for this T010 transaction, however, your application cannot initiate another transaction request until the first one has completed. You can set synchronous or asynchronous mode only with the initialization transaction, and only at the beginning of your session.

Validating Data

The LLAPI does not perform response validation as extensively as do panel dialogs in Tivoli Information Management for z/OS entry or inquiry mode. However, it does provide for some response validation and for some equal sign processing using the = sign. Data from validation records can be used to construct PIPTs and thus be used for validation. You can also define another field in the record that names a validation record to use for validating field data. Your application can use the field validation module BLGPPFVM to validate response data, or you can write your own validation routines using the validation pattern tables provided by the LLAPI. With regard to equal sign processing, if an equal sign is passed as data and the PICA flag PICAEQRP=Y is set, then the API will attempt to process the equal sign using the validation patterns in the PIPT. The four patterns which are currently supported in the API environment are:

- DATE
- TIME
- USER
- CLASS

Collecting Data in Mixed Case

Data which is not validated is passed through the API in the case in which your application supplies it. To convert the data to the case specified in the PIDT (derived from the assisted-entry panel or data attribute record for the field), you must call the validation module described in “Using BLGPPFVM To Validate Data Fields” on page 279.

Loading and Initializing

Your application must establish program linkage to the server routineBLGYSRVR before you can initiate the Tivoli Information Management for z/OS environment. You initiate the Tivoli Information Management for z/OS environment by using the environment initialization transaction (T001). The LLAPI performs all other transactions only after your application initializes the Tivoli Information Management for z/OS environment. The LLAPI environment requires that the MVS/ESA™ operating system, data management services, and VSAM be available. See “Environment Control Transactions” on page 27 for more information on transaction T001.

Recovering from Errors

The LLAPI does not provide error recovery. However, you can design your application to attempt error recovery.

Terminating

To end the Tivoli Information Management for z/OS environment, your application calls the LLAPI using the environment termination transaction (T002). This transaction frees up any resources held by Tivoli Information Management for z/OS.

Your application is then responsible for deleting the server routine. See "Environment Control Transactions" on page 27 for more information on transaction T002.

LLAPI Logic

The LLAPI provides two operating modes:

- the mode which uses some of your interactive panel flow (called *panel processing*)
- the mode which uses none of your interactive panel flow except when processing the delete transaction; this mode is called *bypass panel processing*.

For information on how the LLAPI files records, see page 20. If you write applications that process customized Tivoli Information Management for z/OS records, you might need to perform setup steps or tailor the LLAPI TSPs to correctly process these records. See "Tips for Writing an API Application" on page 273, "Tailoring the Application Program Interfaces" on page 289, and "Terminal Simulator Panels" on page 349 for more information.

Exit and Terminal Simulator Limitations

Program exits, Terminal Simulator Panels (TSPs), and Terminal Simulator Execs (TSXs) that start as a result of file selection are the only program exits, TSPs, and TSXs that run in the interface environment. If you choose to bypass panel processing, file processing is performed by user exit BLGYAPRF and the file control panel is not processed. If you use data model records, you can define in the data view record a TSP or TSX to be run upon record file. Whether you use panel processing or bypass panel processing, program exits, TSPs, and TSXs called during panel processing in interactive mode are *not* called when your application accesses Tivoli Information Management for z/OS. Your application must perform these functions. For example, in the interactive reporter dialog, a program exit can automatically add the reporter's phone number. This program exit is not called when the interface creates a record unless you use data model records. Therefore, the application that creates records through the interface must add the reporter's phone number. You can use some of your existing automation on the create and update transactions by calling program exits and linking to TSPs from the TSPs that control these transactions. Review any commands run by TSPs or TSXs because certain commands cannot run in an API environment (see "Command Limitations" on page 24 for more information).

The value for command processing detection in the user profile determines how an assisted-entry command reply is handled. If you modify or use TSPs or TSXs that enter information into assisted-entry panels, be aware that the LLAPI changes the value of this entry from PROMPT to DATA.

The value for *Quick Search?* in the user profile determines if structured search mode or quick search mode is issued within Tivoli Information Management for z/OS. The LLAPI changes the value for *Quick Search* to YES.

If you use data model records, some program exits can be invoked when data is entered into fields. "Field Validation Using the Field Validation Module BLGPPFVM" on page 279 contains a discussion of data model records and a list of supported program exits.

The following occurs if a TSP or TSX receives a severe error:

- Fields TSCAFRET and TSCAFRES are set to 8.

- The reply buffer is cleared.
- The current dialog is ended.
- The severe error panel is not displayed.

If the severe error generates any messages, they are returned to your application or printed, if your application requests messages to be printed or returned.

Record Update Retry and Wait Considerations

There is a small time window during which a record is unavailable when another user is attempting to check out, update, or check in that record. This could make the record unavailable to you for update even if you have checked out the record. To avoid receiving an error from the LLAPI indicating that the record is unavailable, you can direct the LLAPI to do either of the following:

- Retry the transaction from 1 to 255 times before returning control to the application with the normal message or return code indicating the record is busy.
- Retry the transaction when the record is released.

The transaction can gain access to the record for your application as soon as the record is released by the application or user that is attempting to access the record. For example, you direct the LLAPI to attempt to perform the update record transaction five times. If the first attempt fails, the LLAPI attempts the transaction again. If the record is released before the second attempt, the LLAPI successfully completes the transaction and returns control to your application. The LLAPI performs five retries in all before returning control to your application.

To direct the LLAPI to retry these transactions, ask the person responsible for defining the attributes in this session-parameters member to add the APIENQ=NOWAIT parameter to the BLGPARMS macro. The default number of retry attempts is 25. If you want to specify another number of retry attempts, specify the APIRETRY=N parameter on the BLGPARMS macro, where **N** is the number of retry attempts.

Functions Shipped Disabled

The following LLAPI functions are shipped disabled:

- On the create record (T102) and update record (T105) transactions:
  - Use a dynamic PIDT
  - Process history data
  - Process text audit data
- On the delete record (T110) transaction:
  - Delete a damaged record with the root VSAM key

If you are using panel processing, you can enable these functions by modifying the TSPs for these LLAPI transactions. For the

- Create record transaction, modify TSP BLGAPI02
- Update record transaction, modify TSP BLGAPI05
- Delete record transaction, modify TSP BLGAPI10

If you have chosen to bypass panel processing, you can modify TSP BLGAPIPX to enable those functions shipped disabled. See “Terminal Simulator Panels” on page 349 for information on the TSPs and how to modify them.

Once the functions for a transaction are enabled, the TSP checks for database administration authority.

Multiple Response List Data Item Processing Considerations

A multiple response list item is a list column for which a single entry allows more than 1 word. The LLAPI does not support multiple response list items.

Addressing

Applications using the LLAPI can reside in an address space above or below the 16MB address range. The components of the interface all reside above the 16MB address range. Applications using either 24-bit or 31-bit addressing can call the server.

If your application runs below the 16MB address range, it must transfer control to the server using the MVS LINK macro. Using the LINK macro assures correct address mode maintenance.

The LLAPI allocates storage and returns data using addresses above the 16MB address range if you specify the PICAHMEM=Y when you initialize the LLAPI.

Checking Records In and Out

Checking out a record with an API differs from what interactive users of Tivoli Information Management for z/OS are used to. When you check out a record with an API, it remains checked out and unavailable to anyone else for update until you perform a check in transaction, until an optional administrator-specified time limit is reached, or until an administrator manually checks in the record. This way, you can be sure that the record you want to work with is unchanged from the time you find it until the time you make your own changes to it, even if your application ends before it checks in the record. Your system administrator can define an expiration time that will, in effect, check in records after the specified period of time. See "Check Out Record (T104)" on page 47 and "Check Out Record (HL04)" on page 162 for additional information on this process.

If your application runs a check–out transaction for any record, be sure to check it back in when you finish with it.

Note: If you do not check in a record, the system administrator can check it in interactively. Refer to the *Tivoli Information Management for z/OS User's Guide* for details on database cleanup.

LLAPI Environment Considerations

Your application must call the server BLGYSRVR in problem program state with storage key 8 under the control of a task that was attached with storage key 8. If it does not, unpredictable results can occur, such as an ABEND 0C4.

NetView Considerations

If your application runs under NetView, all Tivoli Information Management for z/OS components must be put in an authorized program facility (APF) library.

Notification Considerations

API processing allows mail notification to users to be performed when an API record is successfully filed. A TSP or TSX can be invoked from file processing if you are using panel processing or you can define in a data view record the name of a TSP or TSX to be invoked when a record is filed. A TSX can send mail via MVS TCP/IP SMTP or local processing. For more information on mail notification and TSPs or TSXs, refer to the *Tivoli Information Management for z/OS Program Administration Guide and Reference*.

Using Alias Names

The LLAPI provides transactions that obtain and free resources for alias tables, but it does not provide alias table processing. Your application must provide the processing.

Record File Processing

If you are using panel processing, the LLAPI performs record file processing for create and update transactions by using Selection 9 (File Record) on summary panels. It processes the record just as if you had used the panel interface. That is, certain data fields, such as Date last altered, Time last altered, and Time entered, are automatically set by Tivoli Information Management for z/OS.

If you are using the bypass panel processing function, record file processing is performed by user exit BLGYAPRF.

Date Considerations

Dates used by your application can be processed in either of two ways:

Database format

Dates are passed to your application from the API in the default external date format. Dates your application passes to the API must be in either the default format or, if one is defined, the old format specified in the session parameters being used. Dates passed in either format are automatically converted to internal format when they are stored in the SDDS portion of the database.

Application-specified format

Dates are passed between the API and your application in a date format your application specifies. This format does not need to match that of the database. The API automatically converts dates from the internal format in the database to the format you specify when passing data to your application and from your specified format to the database's internal format when receiving data from your application.

An application-specified date format is set in the LLAPI by specifying the desired date format in the PICADFMT and PICADSEP fields. If you choose this option and your date format is longer than PIDTMAXL for a field, the entire date will be returned and PIDTCURL will be larger than PIDTMAXL.

The default external format is the default and can be specified in the LLAPI by leaving the PICADFMT field set to zero or setting it to zero if an application-specified format was used for the previous transaction.

Logical Database Partitioning

If you are using logical database partitioning, you can perform the database access transactions (retrieve, update, check in, check out, add record relation, and delete) only for records whose Owning Partition matches the Primary Partition of your privilege class. You should also be aware that API applications cannot perform multipartition searches.

LLAPI Calls

This example shows the interface call syntax which uses call-with-parameter-list notation.

```
<Label> CALL BLGYSRVR (parameter list)
```

Figure 4 shows the parameter list (PLIST) structure used for calling the interface as it appears to an Assembler language program. The parameter list points to the LLAPI communications area (PICA). Your application allocates and initializes the PICA. The PICA cannot be allocated in protected storage.

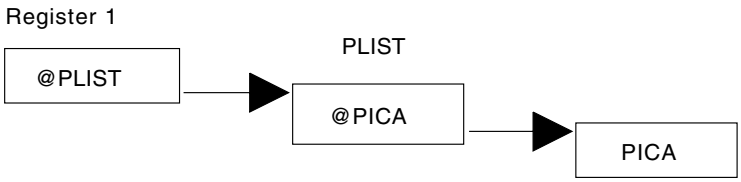

Figure 4. Input Parameter List for the LLAPI

This example shows how to code the setup of the PICA for your application. This is a sample of various sections of code. It is not a complete application.

```
*********************************************************************
*         GETMAIN AREA TO CONTAIN THE PICA                          *
*********************************************************************
         LA    R0,APICALNG           LOAD ADDRESS OF THE PICA LENGTH
         GETMAIN R,LV=(0)            STORAGE FOR PICA
         LR    R5,R1                 GETMAIN FOR PICA WORK AREA
⋮

*********************************************************************
*         INITIALIZE THE PARAMETER LIST                             *
*********************************************************************
         LA    R1,PARMLIST           ADDRESSABILITY TO PARMLIST
         ST    R5,PICAADR            STORE ADDRESS OF PICA
*********************************************************************
*         CALL API INTERFACE                                        *
*********************************************************************
         LINK  EP=BLGYSRVR
⋮

*********************************************************************
*         PARMLIST DEFINED                                          *
*********************************************************************
PARMLIST DS    0F                PLIST FOR
PICAADR  DS    F                 PARM1 (ADDR OF PICA)
⋮

*********************************************************************
*         PICA                                                      *
*********************************************************************
         BLGUPICA
APICALNG EQU   *-PICAACRO         LENGTH OF PICA
⋮
```

Data Sets

The LLAPI uses the following data sets:

- **Text data set**

  This data set stores text data for a unique text record. You use text data sets when:
  - Creating records that contain text
  - Updating records that contain text
  - Retrieving records that contain text

Note: You can process text using a storage buffer and avoid the processing overhead associated with data set manipulation. See "Text Processing Considerations" on page 59 for details.

When retrieving text, the LLAPI creates the data set according to the following conventions:

Data Set Name (DSN) =

- APPL_ID.TEXT_TYPE_S-WORD_INDEX.DYYDDD.THHMMSS.THS.TEXT

APPL_ID (1st qualifier)
The application name in PICA field PICAUSRN.

TEXT_TYPE_S-WORD_INDEX (2nd qualifier)
The s-word index of the text item in the data set.

When using a dynamic PIDT, this qualifier is X*nnnn*, with *nnnn* being the number of the freeform text entry in the record. For example, the first freeform text entry in the record is X*0001*, the second is X*0002*, and so on.

DYYDDD (3rd qualifier)
The character D followed by the Julian date expressed as YYDDD.

THHMMSS (4th qualifier)
The character T followed by the time expressed as HHMMSS.

THS (5th qualifier)
The time extended to tenths and hundredths of a second.

TEXT (6th qualifier)
The word TEXT.

When creating or updating text in the database, you can specify any valid data set name up to 44 characters long.

When you retrieve text (text and audit) data from the database, the contents of the data set are:

- Text Data – 132 bytes of text

  Note: If the freeform text is modified to contain more than 132 bytes of text, only 132 bytes are returned.
- Audit Data – 36 bytes

  Blank (1 byte)
  Date in Julian format (YYDDD) (5 bytes)
  Blank (1 byte)
  Time in format (hh:mm:ss) (8 bytes)
  Blank (1 byte)
  Application or user ID (8 bytes)
  Blank (1 byte)
  Privilege class name if present (8 characters) (8 bytes)
  Blanks (3 bytes)
- DCB Parameters (For both input and output. Retrieving text only uses output.)

Data set organization	= Sequential
Device type	= DASD
Record format	= Fixed block
Blocksize	= 5280 when data set is used for input and audit data is not specified
	= 6216 when data set is used for output and audit data is not suppressed, or when data set is used for input and audit data is specified
	= 6336 when data set is used for output and audit data is suppressed
Record length	= 132 when data set is used for input and audit data is not specified, or when data set is used for output and audit data is suppressed
	= 168 when data set is used for output and audit data is not suppressed, or when data set is used for input and audit data is specified
Contents	= variable text data.

- **Report format table data set**

  This data set contains the data, pattern, and alias tables used by the LLAPI. It also includes report format tables (RFTs) used for report generation. You can access the data set through the session-parameters member named during interface initialization or through the use of the RFTDD DDNAME.

  If you concatenate report format table data sets, all block sizes do not have to be the same.

- **SYSPRINT data set**

  This data set contains, among other things, messages that can result from an abnormal termination of the API subtask. The data set must be a sequential non-VSAM data set you write to a system output device, a tape volume, or a direct access volume.

  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
  ```

  or

  ```
  BLKSIZE - 4
  ```

In all cases, the LRECL must be less than or equal to (BLKSIZE - 4). If this is not the case when the data set is opened, an abend will occur because the data attributes are inconsistent.

The RECFM of the SYSPRINT data set must be VBA and the DSORG must be PS.

- **APIPRINT data set**

  If you set the PICAMSGD field to B or P, this data set contains messages about transaction activity. The data set must be a sequential non-VSAM data set that you write to a system output device, a tape volume, or a direct access volume.

  DCB parameters for this data set are:
  DSORG = PS
  RECFM = VBA
  LRECL = 125
  BLKSIZE = 6144

  If you do not allocate the APIPRINT data set before you request logging, the LLAPI dynamically allocates APIPRINT to a SYSOUT=A data set.

- **SYSMDUMP data set**

  This data set must be defined to receive dump output that can result from an abnormal termination of the API subtask. The data set must be a sequential non-VSAM data set that you write to a tape volume or a direct access volume.

  DCB parameters for this data set are:
  DSORG = PS
  RECFM = FB
  LRECL = 4160
  BLKSIZE = 4160

LLAPI Considerations and Restrictions

You must use unique s-words or p-words for all data responses to define the context of their use, and you must define the field as *replaceable*. The term replaceable means that when the LLAPI collects a unique field response, the last response collected for the field replaces all previously collected responses for the field when the LLAPI stores the response in the database. Refer to the *Tivoli Information Management for z/OS Panel Modification Facility Guide* for more information on s-words.

The following transactions are exceptions to this limitation:

- Add record relation
- Create using dynamic PIDTs
- Update using dynamic PIDTs

Responses collected using the add record relations transaction are marked as nonreplaceable. Responses collected using dynamic PIDTs are marked replaceable or nonreplaceable depending on how the data is stored in the record that was retrieved to create the dynamic PIDT.

The APIs enable your application to access Tivoli Information Management for z/OS without passing SRCs in a batch-type panel driven process. The APIs do not support SRC records.

Command Limitations

When the LLAPI initializes Tivoli Information Management for z/OS,the following Tivoli Information Management for z/OS commands are disabled:

- CHANGE
- DRAW
- DROP
- ISPF
- PRINT
- RECALL
- REPORT
- WINDOW

If any user-written TSP or TSX issues such a command, Tivoli Information Management for z/OS issues message BLG03046W, "The specified response is not a command or a panel reply."

Errors and Messages

Error conditions detected in the LLAPI are associated with a unique transaction. Return codes indicate the transaction's success or failure.

The LLAPI can issue messages to the APIPRINT activity log. The LLAPI also provides a way to pass Tivoli Information Management for z/OS messages to the calling application by using the interface message chain. When returning messages on the chain, PICA field PICAMSGP contains a pointer to the message chain. PICA field PICAMSGC contains a count of the messages on the chain. The messages on the chain are variable length.

The API subtask provides messages about the LLAPI transactions and functions. The API writes these messages to the activity log or places them on the message chain or both. Refer to *Tivoli Information Management for z/OS Messages and Codes* for messages issued by the LLAPI. See "Return and Reason Codes" on page 301 for return codes and reason codes issued by the LLAPI.

Structures

For detailed information on the LLAPI structures and their fields, refer to the sections in the following list:

- Program interface communications area (PICA), see "Low-Level Program Interface Communications Area (PICA)" on page 101.
- Program interface alias table (PALT), see "Program Interface Alias Table (PALT)" on page 112.
- Program interface data table (PIDT), see "Program Interface Data Table (PIDT)" on page 114.
- Program interface history table (PIHT), see "Program Interface History Table (PIHT)" on page 132.
- Program interface pattern table (PIPT), see "Program Interface Pattern Table (PIPT)" on page 136.
- Program interface argument table (PIAT), see "Program Interface Argument Table (PIAT)" on page 139.
- Program interface results table (PIRT), see "Program Interface Results Table (PIRT)" on page 141.
- Program interface message block (PIMB), see "Program Interface Message Block (PIMB)" on page 143.

LLAPI Transactions

The LLAPI transactions your application uses are divided into three groups:

- Environment control
- Service
- Database access

Your application uses the environment control transactions to establish and to close the Tivoli Information Management for z/OS environment and the LLAPI. Your application uses the service transactions to access services, such as storage allocation and deallocation for other transactions, obtaining special tables, checking in and checking out records being modified by the LLAPI, and checking on the progress of transactions in process by the LLAPI. Your application uses database access transactions to perform the database tasks listed at the beginning of this chapter on page 15. Table 2 lists each function that the LLAPI performs, its associated transaction number, and the page in this book where you can find more information about the function.

Table 2. LLAPI Functions and Transactions

LLAPI Function	Transaction Number	page
Initialize Tivoli Information Management for z/OS	T001	27
Terminate Tivoli Information Management for z/OS	T002	30
Obtain external record ID	T003	31
Obtain pattern table (PIPT)	T004	33
Free pattern table (PIPT)	T005	34
Free data table (PIDT)	T006	35
Free result table (PIRT)	T007	36
Check in a record	T008	37
Sync and wait on completion	T009	39
Check transaction completion	T010	40
Obtain alias table (PALT)	T011	41
Free alias table (PALT)	T012	42
Load PIDT	T013	43
Retrieve record	T100	55
Obtain record create resource	T101	44
Create record	T102	63
Obtain record update resource	T103	45
Check out a record	T104	47
Update record	T105	73
Obtain inquiry resource	T106	49
Record inquiry	T107	84
Obtain add record relation resource	T108	50
Add record relation	T109	93
Delete record	T110	96
Start user TSP	T111	52

Table 2. LLAPI Functions and Transactions (continued)

LLAPI Function	Transaction Number	page
Change record approval	T112	98

The remainder of this chapter describes the use of these transactions. For each transaction, introductory text describes required and optional structure (control blocks, tables) fields, their value settings, and their relationships to other structures. The descriptions include tables that show transaction flow from the application through the LLAPI and back to the application.

Environment Control Transactions

Use this group of transactions to initialize and end an LLAPI environment. You can also establish particular operating characteristics for the environment. The environment control transactions are T001 and T002.

Initialize Tivoli Information Management for z/OS (T001)

This transaction initializes the Tivoli Information Management for z/OS environment and prepares the LLAPI for other transaction processing. It also lets you establish particular environment operating characteristics by setting values in various fields in the PICA. Your application can initialize any number of environments to run concurrently, but you must save each one in a unique PICA or application pointer variable.

Each time you initialize a Tivoli Information Management for z/OS session, you establish a storage environment for that particular instance of Tivoli Information Management for z/OS. PICA field PICAENVP points to this storage environment.

Follow these steps to initialize Tivoli Information Management for z/OS:

1. Define storage areas for each PICA control block your application uses.
2. Initialize PICA fields to govern how this session instance of the LLAPI is to operate. If you choose to use bypass panel processing, you must set PICADRIF=Y at initialization.
3. Initialize PICAENVP to zero.
4. Initialize PICALENG to the length of the PICA structure.
5. Start the server module BLGYSRVR, passing a parameter list with the first pointer in the parameter list pointing to the PICA.

Your application must set the following PICA fields to the values specified and must maintain the first three of these values throughout the environment's session:

PICAACRO Uppercase character string of PICA.

PICALENG Length of PICA structure in fixed binary format.

PICAENVP A pointer to the address of the Tivoli Information Management for z/OS environment. You must initialize this field to zero through the initialize Tivoli Information Management for z/OStransaction (T001).

PICATRAN A transaction code of T001.

PICAUSRN The name by which Tivoli Information Management for z/OS recognizes your application. Tivoli Information Management for z/OSuses this name in

place of a TSO user ID when performing record access privilege class processing. The value specified must be an eligible user of the initial privilege class record.

PICACLSN A valid privilege class name. This privilege class is known as the initial privilege class.

PICASESS A session-parameters member name. The attributes defined in this session-parameters member initialize Tivoli Information Management for z/OS in a way that is similar to the initialization of an interactive session.

The following PICA fields are optional, and they further define the operating characteristics of the optional fields of the Tivoli Information Management for z/OS session.

PICAASYN Set this field to Y if the server is to operate in asynchronous mode. When the server operates asynchronously, control returns to your application right after the LLAPI validates transaction parameters and the API subtask starts. When using this mode, your application can issue a sync and wait on completion (T009) or check transaction completion (T010) transaction to inquire about the status of the currently processing transaction. Asynchronous mode provides a way for your application to do other work while the API processes your transaction.

PICACLSC A count of the number of privilege class records that can be maintained in storage during this session. Any value of zero or larger is valid. However, if you specify a value of zero, Tivoli Information Management for z/OS uses a value of one. In an interactive session, your application operates under a unique privilege class. If you want to change that class, you must choose a new class from a list of class names you are entitled to use. Each time the user chooses a new class, the LLAPI brings the class record into storage where it is made the new class. In the API's environment, the class records stay in storage as long as there is space for them. This count field specifies the number of class records that can be in storage simultaneously. When storage contains the PICACLSC number of class records and you need a new record, the LLAPI removes the least recently used class record from storage to make room for the new record. If your application uses a large number of different class records, make this count high for improved performance. The value used, of course, must be tempered by considering storage availability.

Note: The API also provides a method (PICACLSN) whereby you can start database access transactions with a different privilege class.

PICATINT Transaction processing time interval. This field specifies the time (in seconds) that any transaction can process before the LLAPI notifies your application of a timeout.

It is important to realize that, because of external loads on the database, there is no guarantee that transactions will run within a certain time. To avoid problems, assign a worst-case time value that your application can endure before timeout occurs and a recovery process must be started.

No transaction recovery can be made for environments operating in synchronous mode. However, if your environment is asynchronous, you can issue a check transaction completion (T010) transaction to restart the timeout interval and obtain the status of the transaction for your application. You can

also issue a sync and wait on completion (T009) transaction to transfer to the interface and wait for the completion of the processing transaction or its timeout. If a timeout occurs, you can issue another T009 or T010 transaction to restart the timeout interval and wait for completion of the processing transaction. In all of these actions your application should check return and reason codes to determine the most appropriate course of action.

PICASPLI Spool interval indicating the number of minutes that the activity log can print transaction results before the LLAPI closes and reopens the log. Closing and reopening the log destroys data previously recorded in the log.

The maximum number of minutes you can use is 60*24 (that is, 60 minutes multiplied by 24 hours=1440 minutes, one full day). If you specify more minutes than there are in a day, the activity log closes and reopens after 1440 minutes, ignoring your specification.

Note: The activity log is intended for debugging and should not serve as a history file.

PICAMSGD A 1-character field indicating destination of messages produced by the API subtask. The character options and their meanings are:

P Returns output messages to an APIPRINT data set
C Returns output messages on the message chain
B Performs the functions of both P and C

Any characters other than P or B are treated as C. Your application sets this field.

PICADBID This field defaults to 5, which indicates access to the Tivoli Information Management for z/OS database. If you have other Tivoli Information Management for z/OS type databases with other identifiers, you can access another database by specifying its ID in this field.

PICAHMEM Set this field to **Y** if you want to allow the LLAPI and Tivoli Information Management for z/OS to allocate memory above the 16MB address range.

PICADRIF Bypass panel processing indicator. This is set by your application. A **Y** indicates that no panels should be used in record processing. Any other value indicates panels should be used. If you specify a value of **Y**, you must also use data model records (**PICADMRC=Y**)if you are using file processing commands (create, update, or add record relation).

Table 3 shows the initialize Tivoli Information Management for z/OS transaction flow for a synchronous environment. T001 starts the LLAPI and prepares Tivoli Information Management for z/OSfor further transaction processing.

Table 3. LLAPI Transaction T001. Initialize Tivoli Information Management for z/OS (Synchronous)

Step	Program	Action
1	Application	■ Establishes linkage to module BLGYSRVR and saves its address ■ Gets storage for a PICA ■ Sets PICA fields as follows: • PICAACRO=PICA • PICALENG=length of PICA • PICATRAN=T001 (Initialize Tivoli Information Management for z/OS) • PICAUSRN=application ID • PICACLSN=privilege class name • PICACLSC=class record count • PICASESS=session-parameters member name • PICAENVP=0 • PICAASYN=blank **Note:** Specify a **Y** in this field to initialize Tivoli Information Management for z/OS in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies a received transaction's validity. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to finish. See explanations of T009 sync and T010 check transactions, page 39. • PICATINT=300 (seconds) ■ Calls BLGYSRVR(PICA).
2	Server	■ Validates PICA fields ■ Attaches API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAENVP **Note:** Your application must maintain the environment block pointer (PICAENVP) until you end the LLAPI session. • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAMSGC contains message count. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Terminate Tivoli Information Management for z/OS (T002)

This transaction stops the LLAPI and terminates the Tivoli Information Management for z/OS environment. You must specify the following PICA field to start this transaction:

PICATRAN A transaction code of T002.

Table 4 shows the terminate Tivoli Information Management for z/OS transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 4. LLAPI Transaction T002. Terminate Tivoli Information Management for z/OS (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T002 (Terminate Tivoli Information Management for z/OS) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 sync and T010 check transactions, page 39. ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Detaches API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAENVP • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAENVP contains 0. • PICAMSGC contains 0. • PICAMSGP contains 0. ■ Continues processing.

Interface Service Transactions

These transactions provide unique services to your application and other transactions. These services include obtaining and freeing storage resources for other transactions and tables, obtaining special tables, checking records in and out, and checking transaction progress. These transactions are T003 through T013, T101, T103, T106, T108, and T111.

Obtain External Record ID (T003)

This transaction obtains an external record ID for use in creating records. On return to your application, the external record ID is stored in PICA field PICARNID. This transaction is useful in providing your application with a centralized record numbering service to prevent duplicate record IDs. Once you obtain the record ID, you cannot return it to the system.

To create a record with this record ID, specify this record ID for the field associated with p-word RNID/ on the create transaction (T102). Be aware that this record ID might not pass validation because record IDs are usually not allowed to be all-numeric.

You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T003.

You can specify values for these PICA fields if you want to change the name of the current application ID or the name of the current privilege class:

PICAUSRN The name by which Tivoli Information Management for z/OS recognizes your application.You can specify a value here to change the name of the current application ID. Tivoli Information Management for z/OS uses this name in place of a TSO user ID when performing record access privilege class processing. The value specified must be an eligible user of the current privilege class record.

PICACLSN A valid privilege class name.You can specify a value here to change the current privilege class record.

Table 5 shows the obtain external record ID transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 5. LLAPI Transaction T003. Obtain External Record ID (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T003 (Obtain External Record ID) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 sync and T010 check transactions, page 39. ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICARNID • PICAMSGC • PICAMSGP ■ Returns to application.

Table 5. LLAPI Transaction T003 (continued). Obtain External Record ID (Synchronous)

Step	Program	Action
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code • PICAREAS contains reason code • PICARNID contains external record ID if transaction is successful • PICAMSGC contains number of messages • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Obtain Pattern Table (T004)

This transaction obtains an existing Program Interface Pattern Table (PIPT) associated with a particular PIDT. To conserve resources, the LLAPI does not obtain a PIPT when it obtains an associated PIDT. You obtain the PIPT only when you want to validate data responses before storing them in a response buffer. This transaction obtains the companion PIPT specified in the PIDT so the application can provide a level of response validation similar to that provided by the assisted-entry panel of an interactive session. The address of the PIPT is stored in PIDT field PIDTPIPT.

The LLAPI allows your application to provide response validation by invoking the BLGPPVFM field validation module or any user-written validation module. Your application must start this transaction to obtain the pattern table before you can use a validation module. You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T004

PICAPIDT Pointer to the PIDT for which this transaction obtains a PIPT

Table 6 shows the obtain PIPT transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see “LLAPI Structures” on page 100.

Table 6. LLAPI Transaction T004. Obtain PIPT (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T004 (Obtain PIPT) • PICAPIDT=address of the PIDT naming the PIPT **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. ■ Calls BLGYSRVR(*PICA*).

Table 6. LLAPI Transaction T004 (continued). Obtain PIPT (Synchronous)

Step	Program	Action
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following fields: • PICARETC • PICAREAS • PIDTPIPT • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PIDTPIPT points to pattern table (PIPT). • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Free Pattern Table (T005)

This transaction frees the storage associated with a particular PIPT. You must specify the following fields to start this transaction:

PICATRAN A transaction code of T005

PICAPIDT Pointer to the PIDT containing the address of the PIPT to be freed.

Table 7 shows the free PIPT transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see on page 39.

Table 7. LLAPI Transaction T005. Free PIPT (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T005 (Free PIPT) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICAPIDT=address of PIDT pointing to PIPT. ■ Calls BLGYSRVR(*PICA*).

Table 7. LLAPI Transaction T005 (continued). Free PIPT (Synchronous)

Step	Program	Action
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following fields: • PICARETC • PICAREAS • PIDTPIPT • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PIDTPIPT contains 0. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Free Data Table (T006)

This transaction frees the storage associated with a particular PIDT. The storage resources include:

- Response buffer storage associated with the PIDT
- PIAT storage associated with the PIDT
- PIPT storage associated with the PIDT
- PIHT storage associated with the PIDT
- PIDT storage

You must specify the following PICA fields to start this transaction:
PICATRAN A transaction code of T006
PICAPIDT Pointer to the PIDT to free

Table 8 shows the free PIDT transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see “LLAPI Structures” on page 100.

Table 8. LLAPI Transaction T006. Free PIDT (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T006 (Free PIDT) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICAPIDT=address of PIDT to free. ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAPIDT • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAPIDT contains 0. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Free Result Table (T007)

This transaction frees the storage associated with a PIRT that the API generates as a result of an inquiry transaction. If you save a search results list, you can specify a search ID (PICASRID), a PIRT address, or both to free all storage for that search. The process used by this transaction to free storage depends on whether you specify a search ID:

- If you specify a search ID, the search results list is freed, and the specified PIRT address is set to zero if the following conditions are true:
  - A search results list is found with that search ID.
  - The specified PIRT address matches the PIRT address of the search results list.
- If you do not specify a search ID and the specified PIRT is part of a search results list, all storage for the search results list is freed, including the specified PIRT.

When the search is freed, it cannot be used to return match results. If successive inquiry transactions require larger PIRTs, the API allocates them. Your application can conserve storage by freeing the PIRT after your application performs its last inquiry transaction.

You must specify the following PICA field to start this transaction:

PICATRAN A transaction code of T007

The following PICA fields are optional, but at least one must be specified:
PICAPIRT Pointer to the PIRT to free.
PICASRID A search ID to free

Table 9 shows the free PIRT transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 9. LLAPI Transaction T007. Free PIRT

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T007 (free PIRT) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICAPIRT=address of PIRT (if PICASRID not specified) • PICASRID=search ID (if search results saved) ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAMSGC • PICAMSGP ■ If PICASRID is not specified or is specified and represents a search results list, sets the following PICA fields to zero: • PICASRID • PICAPIRT ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAPIRT contains 0. • PICASRID contains 0. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Check In Record (T008)

This transaction removes the checkout indicator in a record when the application ID stored in the record is the same as the application ID issuing the transaction request. If another

Tivoli Information Management for z/OS user is attempting to update the record when you attempt to check in the record and the API returns an unavailable condition, your application should restart the check in record transaction as described below until it succeeds.

You can direct the LLAPI to either retry this transaction from 1 to 255 times before returning control to your application or wait until the record is available. See page 18 for more information.

Note: If you are using logical database partitioning, you can check in a record only if the Owning Partition of that record matches the Primary Partition of your privilege class.

You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T008

PICARNID The record ID or root VSAM key for which the checkout indicator is to be removed so other database users can access the record.

You must also specify PICAVSAM=Y in the PICA when using a root VSAM key in PICARNID.

You can specify values for these PICA fields if you want to change the name of the current application ID or the name of the current privilege class:

PICAUSRN The name by which Tivoli Information Management for z/OS recognizes your application.You can specify a value here to change the name of the current application ID. Tivoli Information Management for z/OS uses this name in place of a TSO user ID when performing record access privilege class processing. The value specified must be an eligible user of the current privilege class record.

PICACLSN A valid privilege class name.You can specify a value here to change the current privilege class record.

Table 10 on page 39 shows the check in record transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see “LLAPI Structures” on page 100.

Table 10. LLAPI Transaction T008. Check In Record (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T008 (Check In Record) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICARNID=record ID or root VSAM key of record to check in • PICAVSAM=Y if using a root VSAM key. • PICAUSRN=application ID if you want to change the name of the current application. • PICACLSN=privilege class if you want to change the privilege class. ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Sync and Wait On Completion (T009)

This transaction allows your application to wait for a previously attempted transaction to complete. When the wait ends, the LLAPI provides information about the previously attempted transaction. If the transaction is not completed when the sync and wait transaction is issued, the API returns control to the application when the previously attempted transaction either completes or ends with a time-out. If a transaction timeout interval occurs for the sync and wait transaction, the application can issue another sync (T009) or check (T010) transaction to initiate another time interval and continue the sync and wait transaction. The application can also issue the terminate transaction (T002).

The PICA return and reason codes correspond to the previously attempted transaction. The codes returned by the API depends on the condition detected by the API. The LLAPI returns a warning return code with a reason code that indicates no transaction is active. The LLAPI returns various other codes to indicate transaction completion. See “Return and Reason Codes” on page 301 for more information about codes that can be returned in the PICA.

You must specify the following PICA field to start this transaction:

PICATRAN A transaction code of T009.

Table 11 shows the sync and wait on completion transaction flow for an asynchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 11. LLAPI Transaction T009. Sync and Wait on Completion (Asynchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T009 (sync and wait on completion) ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Check Transaction Completion (T010)

This transaction provides your application with the status of a previously attempted transaction. If the transaction is not finished when you start the check transaction, the API immediately returns control to the application and provides the status of the previous function. This transaction does not schedule any work by the API task.

The PICA return and reason codes correspond to the previously attempted transaction. The codes returned by the API depends on the condition detected by the API. The LLAPI returns a warning return code with a reason code that indicates no transaction is active. The LLAPI returns various other codes to indicate transaction completion. See "Return and Reason Codes" on page 301 for more information about codes that can be returned in the PICA.

You must specify the following PICA field to start this transaction.

PICATRAN A transaction code of T010.

Table 12 on page 41 shows the check transaction completion transaction flow for an asynchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 12. LLAPI Transaction T010. Check Transaction Completion (Asynchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T010 (Check Transaction Completion) ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Checks previous transaction's status ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Obtain Alias Table (T011)

This transaction retrieves a specified alias table (PALT) from its PDS, and creates a copy of it for the LLAPI to use. You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T011
PICATBLN The name of the table to retrieve

Table 13 shows the obtain alias table transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 13. LLAPI Transaction T011. Obtain Alias Table (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T011 (Obtain Alias Table PALT) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICATBLN=requested alias table name ■ Calls BLGYSRVR(*PICA*).

Table 13. LLAPI Transaction T011 (continued). Obtain Alias Table (Synchronous)

Step	Program	Action
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICATBLP • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICATBLP points to alias table. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Free Alias Table (T012)

This transaction frees the allocated storage of an alias table (PALT). You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T012
PICATBLP A pointer to the alias table to be freed.

Table 14 shows the free alias table transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 14. LLAPI Transaction T012. Free Alias Table (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T012 (Free Alias Table PALT) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICATBLP=alias table to be freed. ■ Calls BLGYSRVR(*PICA*).

Table 14. LLAPI Transaction T012 (continued). Free Alias Table (Synchronous)

Step	Program	Action
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICATBLP • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICATBLP contains 0. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Load PIDT (T013)

This transaction is used to load a static PIDT (or generate a PIDT from data model records) to obtain information about the data model for a record. PIDTs obtained via this transaction should not be used on subsequent database access transactions (for example, record create). The value for PIDTREQD is always returned as `N`. You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T013.
PICATABN The name of the static PIDT or data view record.
PICADMRC Specify a value of **Y** if PICATABN contains a data view name; specify a value of blank if PICATABN contains a static PIDT name.
PICAREQL Set to 0.

Table 15 shows the load PIDT transaction flow. For more detailed information on the LLAPI structures and their fields, see “LLAPI Structures” on page 100.

Table 15. LLAPI Transaction T013. Load PIDT

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T013 (Load PIDT) • PICATABN=name of the static PIDT or data view name • PICADMRC=Y to indicate that PICATABN is a data view name or blank to indicate that PICATABN is a static PIDT name • PICAREQL=0 to not obtain a response buffer ■ Calls BLGYSRVR(*PICA*).

Table 15. LLAPI Transaction T013 (continued). Load PIDT

Step	Program	Action
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAPIDT • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAPIDT points to PIDT. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Obtain Record Create Resource (T101)

This transaction constructs table and storage resources for record creation. If you are using static PIDTs built by BLGUT8, the LLAPI allocates storage for a new PIDT, loads the specified PIDT from the Report Format Table data set concatenation, and stores its address in PICA field PICAPIDT. If you are using data model records, the PIDT is built fromthe specified data view record. The LLAPI also allocates storage for a response buffer and stores its address in PIDT field PIDTBUFP. Your application specifies the required size of the response data buffer in PICA field PICAREQL.

You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T101.

PICATABN Name of static record create PIDT name. If you are using data modelrecords (that is, PICADMRC=Y), then PICATABN contains the record ID (RNID) of the data view record.

PICAREQL Requested response buffer length.

If you are using data view records, and an error is returned(PICARETC not equal 0), check PICAPIDT for the address of the PIDT. If one was returned, search for PIDTCODEs to find any additional error codes and you must also free any storage that was obtained.

It can take a significant amount of time to generate a PIDT from data view records. The length of time depends on the number of data attribute records (and validation records that they reference) contained in the data view record. As with any PIDT, you can maintain the PIDT in storage for subsequent use.

Table 16 shows the obtain create-resource transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see “LLAPI Structures” on page 100.

Table 16. LLAPI Transaction T101. Obtain Create-Resource (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T101 (Obtain Create-Resource) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICATABN=record-create PIDT name. If you are using data models (**PICADMRC=Y**), then this is the data view name. • PICAREQL=requested response buffer length ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following fields: • PICARETC • PICAREAS • PICAPIDT • PIDTBUFP • PIDTBUFL • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAPIDT points to PIDT. • PIDTBUFP points to response buffer. • PIDTBUFL contains length of response buffer. • PIDTCODE contains any field error codes. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Obtain Record Update Resource (T103)

This transaction constructs table and storage resources needed for record update. If you are using static PIDTs built by BLGUT8, the LLAPI allocates storage for a new PIDT, loads the specified PIDT from the Report Format Table data set concatenation, and stores its address in PICA field PICAPIDT. If you are using data model records, the PIDT is built from the specified data view record. The LLAPI also allocates storage for a response buffer and stores its address in PIDT field PIDTBUFP. Your application specifies the required size of the response data buffer in PICA field PICAREQL.

You must specify the following PICA fields to start this transaction:
PICATRAN A transaction code of T103.

PICATABN Update resource PIDT name. If you are using data model records (that is, PICADMRC=Y), then PICATABN contains the record ID (RNID) of the data view record.

PICAREQL Requested response buffer length.

If you are using data view records, and an error is returned (PICARETC not equal 0), check PICAPIDT for the address of the PIDT. If one was returned, search for PIDTCODEs to find any additional error codes and you must also free any storage that was obtained.

It can take a significant amount of time to generate a PIDT from data view records. The length of time depends on the number of data attribute records (and validation records that they reference) contained in the data view record. As with any PIDT, you can maintain the PIDT in storage for subsequent use.

Table 17 shows the obtain update-resource transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 17. LLAPI Transaction T103. Obtain Update Resource (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T103 (Obtain Update Resource) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICATABN=record update PIDT name. If you are using data models (**PICADMRC=Y**), then this is the data view name. • PICAREQL=requested response buffer length ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following fields: • PICARETC • PICAREAS • PICAPIDT • PIDTBUFP • PIDTBUFL • PICAMSGC • PICAMSGP ■ Returns to application.

Table 17. LLAPI Transaction T103 (continued). Obtain Update Resource (Synchronous)

Step	Program	Action
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAPIDT points to PIDT. • PIDTBUFP points to response buffer. • PIDTBUFL contains buffer length. • PIDTCODE contains any field error codes. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Check Out Record (T104)

This transaction provides a mechanism for the LLAPI to hold a record for update. The check out function automatically updates the record and adds an indicator to prevent subsequent record updates by another user or application. This does not prevent other users from attempting to access the record; it only prevents them from updating the record. Any transactions attempting to update the record might not be able to access the record immediately and might have to try one or more times. The LLAPI does not allow multiple check outs of the same record by the same application ID. Record check out indicators are removed by any of the following actions:

- Your application performs a Check In Record (T008) transaction

  If another Tivoli Information Management for z/OS user is attempting to update the record when you attempt to check out the record and the API returns an unavailable condition, your application should restart the check out record transaction as described below until it succeeds.

  You can direct the LLAPI to either retry this transaction from 1 to 255 times before returning control to your application or wait until the record is available. See page 18 for more information.
- The expiration timeout limit is exceeded.
- The database administrator performs an interactive check in of the record.

Check out transactions store the ID of the current application in the record. This provides a mechanism to track and administer record check outs. When an application checks out a record, the record is unavailable for update by anyone else (either other API applications or interactive users) until it is checked in.

Note: If you are using logical database partitioning, you can check out a record only if the Owning Partition of that record matches the Primary Partition of your privilege class.

In order to reduce the risk of leaving a record indefinitely in checked–out status, you may wish to specify the BLX-SP parameter APICHKOUTLIM (this is described in greater detail in the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*). When a check–out limit is specified, the check–out record process reads the value specified for this parameter and performs one of the following functions:

- If the record is not already checked out, or it is checked out to a different application ID and the check out time has expired, the record is checked out to the new application ID and the check out time period is added to the current clock time and stored in the record.

- If the record is already checked out to a different application ID and the check out time has not expired, an error is returned indicating that the record is in use.
- If the record is already checked out to the same application ID, then the expiration time is reset to a full check out time period and saved in the record.

The expiration time is also checked by the Update Record (T105) transaction, by the Add Record Relations (T109) transaction, by the Delete Record (T110) transaction, and by interactive update and delete processing.

You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T104

PICARNID External record ID or root VSAM key. You must also specify PICAVSAM=Y in the PICA when using a root VSAM key in PICARNID.

You can specify values for these PICA fields if you want to change the name of the current application ID or the name of the current privilege class:

PICAUSRN The name by which Tivoli Information Management for z/OS recognizes your application.You can specify a value here to change the name of the current application ID. Tivoli Information Management for z/OS uses this name in place of a TSO user ID when performing record access privilege class processing. The value specified must be an eligible user of the current privilege class record.

PICACLSN A valid privilege class name.You can specify a value here to change the current privilege class record.

Table 18 shows the check out record transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 18. LLAPI Transaction T104. Check Out Record (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T104 (Check Out Record) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICARNID=record ID or root VSAM key of record to check out • PICAVSAM=Y if using a root VSAM key. • PICAUSRN=application ID if you want to change the name of the current application. • PICACLSN=privilege class if you want to change the privilege class. ■ Calls BLGYSRVR(*PICA*).

Table 18. LLAPI Transaction T104 (continued). Check Out Record (Synchronous)

Step	Program	Action
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Obtain Inquiry Resource (T106)

This transaction constructs table and storage resources for record inquiry transactions. If you are using static PIDTs built by BLGUT8, the LLAPI allocates storage for a new PIDT, loads the specified PIDT from the Report Format Table data set concatenation, and stores its address in PICA field PICAPIDT. If you are using data model records, the PIDT is built from the specified data view record. The API allocates storage for a response buffer and stores its address in the PIDT field PIDTBUFP. Your application specifies the required amount of response buffer storage in PICAREQL.

Structured inquiry arguments (stored in the PIDT) simulate quick-search responses. The LLAPI allocates the PIAT needed to specify freeform search arguments. Each freeform argument word occupies an individual PIAT row. Your application specifies the number of rows needed.

You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T106.

PICATABN Record inquiry PIDT name. If you are using data model records (that is, PICADMRC=Y), then PICATABN contains the record ID (RNID) of the data view record.

PICAREQL Requested response buffer length (must be greater than 0)

PICAREQR Number of PIAT rows.

If you are using data view records, and an error is returned (PICARETC not equal 0), check PICAPIDT for the address of the PIDT. If one was returned, search for PIDTCODEs to find any additional error codes and you must also free any storage that was obtained.

It can take a significant amount of time to generate a PIDT from data view records. The length of time depends on the number of data attribute records (and validation records that they reference) contained in the data view record. As with any PIDT, you can maintain the PIDT in storage for subsequent use.

Table 19 shows the obtain inquiry resource transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 19. LLAPI Transaction T106. Obtain Inquiry Resource (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T106 (Obtain Inquiry Resource) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICATABN=inquiry PIDT name. If you are using data models (**PICADMRC=Y**), then this is the data view name. • PICAREQL=request response buffer length • PICAREQR=number of PIAT rows ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following fields: • PICARETC • PICAREAS • PICAPIDT • PIDTBUFP • PIDTBUFL • PIDTPIAT • PICAMSGC • PICAMSGP • PIATNUMR ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAPIDT points to PIDT. • PIDTBUFP points to response buffer. • PIDTBUFL contains buffer length. • PIDTCODE contains any field error codes. • PIDTPIAT points to PIAT. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. • PIATNUMR contains the number of rows allocated in the PIAT. ■ Continues processing.

Obtain Add Record Relation Resource (T108)

This transaction constructs table and storage resources required for adding record relations to parent records. If you are using static PIDTs built by BLGUT8, the LLAPI allocates storage

for a new PIDT, loads the specified PIDT from the Report Format Table data set concatenation, and stores its address in PICA field PICAPIDT. If you are using data model records, the PIDT is built from the specified data view record. The LLAPI allocates storage for an add record relations PIDT, loads the specified PIDT from the report format table data set concatenation, and stores its address in PICA field PICAPIDT. It also allocates storage for a response buffer and stores its address in PIDT field PIDTBUFP. Your application specifies the size of the response buffer storage required in PICA field PICAREQL.

You must specify the following PICA fields to start this transaction:

PICATRAN Transaction code of T108

PICATABN Record relation PIDT name. If you are using data model records (that is, PICADMRC=Y), then PICATABN contains the record ID (RNID) of the data view record.

PICAREQL Requested response buffer length.

If you are using data view records, and an error is returned (PICARETC not equal 0), check PICAPIDT for the address of the PIDT. If one was returned, search for PIDTCODEs to find any additional error codes and you must also free any storage that was obtained.

It can take a significant amount of time to generate a PIDT from data view records. The length of time depends on the number of data attribute records (and validation records that they reference) contained in the data view record. As with any PIDT, you can maintain the PIDT in storage for subsequent use.

Table 20 shows the obtain add record relation resource transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 20. LLAPI Transaction T108. Obtain Add Record Relation Resource (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T108 (Obtain Add Record Relation Resource) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICATABN=add record relation PIDT name. If you are using data models (**PICADMRC=Y**), then this is the data view name. • PICAREQL=request response buffer length ■ Calls BLGYSRVR(*PICA*).

Table 20. LLAPI Transaction T108 (continued). Obtain Add Record Relation Resource (Synchronous)

Step	Program	Action
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following fields: • PICARETC • PICAREAS • PICAPIDT • PIDTBUFP • PIDTBUFL • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAPIDT points to PIDT. • PIDTBUFP points to response buffer. • PIDTBUFL contains buffer length. • PIDTCODE contains any field error codes. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Start User TSP or TSX (T111)

Use this transaction to start a user Terminal Simulator Panel (TSP) or Terminal Simulator Exec (TSX).

When your application runs this transaction, the API subtask router panel BLGAPI00 (if you are using panel processing) or the API subtask router panel BLGAPIDI (if you are bypassing panel processing) uses user exit BLGYITSP to invoke a TSP or TSX specified by your application, or, if your application does not specify the name of a TSP or TSX, branches to the label LINKT111.

Note: If your application does not specify the name of a TSP or TSX to invoke, prior to performing this transaction you must have defined a LINK control line to link BLGAPI00 or BLGAPIDI to the user TSP or TSX.

Your application can pass parameter data to the TSP by allocating a user-defined structure and storing its address in PICA field PICAPARM. When the TSP runs, the Terminal Simulator Communications Area (TSCA) field TSCAUPTR contains this address. You can store any type of user data in the parameter structure. Your application can also specify the privilege class that it wants the TSP to run under.

The PICA field PICAPARM can also be used to contain the address of a string to be passed to an invoked TSP in the variable data area or an invoked TSX as an argument. The maximum length of the string, in characters, is 255.

The PICA field PICAPARL signals whether PICAPARM is the address of a user buffer or the address of a string. If PICAPARL is set to 0, then PICAPARM is the address of the user buffer to be passed, and thus TSCAUPTR is set to the address contained in PICAPARM. If

PICAPARL is greater than 0 (to a maximum of 255), then it indicates that PICAPARM is the address of a string; the value of PICAPARL is the length in characters of the address string being passed. A specified string parameter will only be passed as an argument to a TSP or TSX specified in PICAUTSP. A TSP or TSX defined in BLGAPI00 or BLGAPIDI can only be passed a user-specified pointer in TSCAUPTR (the value of PICAPARM).

Note: Setting PICAPARM to the address of a string and PICAPARL to the length of the string is the only way to pass a string parameter to a TSP or TSX specified in PICAUTSP.

The LLAPI imposes certain product command restrictions. For this reason, existing user-written TSPs or TSXs might not run correctly when started from the LLAPI. For more information about these restrictions, see "Command Limitations" on page 24, "LLAPI Considerations and Restrictions" on page 24, and "Exit and Terminal Simulator Limitations" on page 17.

The LLAPI returns any messages generated by the user TSP to the message chain pointed to by the PICA in the same way as all other transactions.

You must specify the following PICA field to start this transaction:

PICATRAN Transaction code of T111.

Specify a value for the following to define the TSP or TSX to be invoked:

PICAUTSP The name of a TSP or a TSX to be invoked. A string of 255 characters can be passed to the TSP (in the variable data area) or TSX (as an argument) by storing the address of the string in PICAPARM and the length of the string in PICAPARL. If you do specify a blank value (X'40') in PICAUTSP, any address specified in PICAPARM is passed using TSCAUPTR to the TSP or TSX specified in the API TSP BLGAPI00 or BLGAPIDI.

You can specify values for these PICA fields if you want to pass data to the TSP or TSX:

PICAPARM Address of a user structure or address of character string of 1 to 255 characters (if specifying PICAPARM greater than 0 and a value in PICAUTSP). If PICAPARL is set to X'00', then PICAPARM is treated as the address of a user buffer. If PICAPARL is a value other than X'00', than PICAPARM is treated as the address of the string of data.

PICAPARL If this has a value of X'00', then PICAPARM is treated as the address of a user buffer to be placed in TSCAUPTR; if this has a value other than X'00', then PICAPARM is treated as the address of the string of data and PICAPARL is the length in characters of the string being passed. This value is ignored if PICAUTSP contains blanks or X'00'.

You can specify values for these PICA fields if you want to change the name of the current application ID or the name of the current privilege class:

PICAUSRN The name by which Tivoli Information Management for z/OS recognizes your application.You can specify a value here to change the name of the current application ID. Tivoli Information Management for z/OS uses this name in place of a TSO user ID when performing record access privilege class processing. The value specified must be an eligible user of the current privilege class record.

PICACLSN A valid privilege class name.You can specify a value here to change the current privilege class record.

Table 21 shows the start user TSP transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 21. LLAPI Transaction T111. Start User TSP (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T111 (Start User TSP) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICAUTSP=name of a TSP or TSX to be invoked. A string of up to 255 characters can be passed to the TSP (in the variable data area) or TSX (as an argument) by storing the address of the string in PICAPARM and the length of the string in PICAPARL. • PICAPARM=depending on the setting of PICAPARL, this can be the address of a user structure or else the address of a string to be passed. • PICAPARL=a flag (if equal to 0) to indicate that PICAPARM is the address of a user structure, or a length (if greater than 0) of the character string being passed. PICAPARL is ignored if PICAPARM is 0 or PICAUTSP does not contain the name of a TSP or TSX to invoke. • PICAUSRN=application ID if you want to change the name of the current application. • PICACLSN=privilege class if you want to change the privilege class. ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code • PICAREAS contains reason code • PICAMSGC contains number of messages • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Database Access Transactions

You use this group of transactions to create, update, retrieve, inquire about, add record relations to, delete records, and perform change record approval in the Tivoli Information Management for z/OS database. They are T100, T102, T105, T107, T109, T110, and T112.

Retrieve Record (T100)

This transaction retrieves the Tivoli Information Management for z/OS record requested from the database. The LLAPI loads a static PIDT or builds a dynamic view of the record or generates the PIDT from a data view record when performing record retrieval.

The following list outlines the record retrieval process:

1. Do one of the following:
   Direct the API to use a static PIDT (specify the static PIDT name in PICATABN).
   or
   Direct the API to build a view of the record (indicate dynamic PIDT processing by specifying PICADYNM=Y).
   or
   Direct the API to generate a PIDT using data model records (specify the record ID of a data view record). If you specify a data view record ID, set PICADMRC to Y.
2. Specify, in field PICARNID, the ID or the root VSAM key of the record to retrieve. If you specify the root VSAM key, set PICAVSAM to Y.

   Note: If you are using logical database partitioning, you can retrieve a record only if the Owning Partition of that record matches the Primary Partition of your privilege class.
3. Start transaction T100.
4. Process the record data.
5. Free the PIDT and buffer resources (T006).

Response Processing Considerations

The PIDT specifies how the LLAPI presents record data to your application. You can change the PIDT name from one transaction to the next to create different views of a unique record type to suit your needs. If you are using data model records, you can change the data view record ID from one transaction to the next to create different views of a unique record type to suit your needs. The LLAPI generates a PIDT from the specified data view record and associated data attribute records. The LLAPI stores the PIDT address in PICA field PICAPIDT. To free the PIDT and response buffer, define your application to perform a Free PIDT transaction (T006) after your application processes the record data. If, on subsequent Record Retrieval (T100) transactions, the name of the PIDT pointed to by PICA field PICAPIDT matches the PIDT name specified in PICA field PICATABN, the LLAPI uses the current PIDT for data extraction. The address of the response buffer associated with the PIDT might change when doing this because the amount of data retrieved is variable.

It can take a significant amount of time to generate a PIDT from data view records. The length of time depends on the number of data attribute records (and validation records that they reference) contained in the data view record. As with any PIDT, you can maintain the PIDT in storage for subsequent use.

Unless you request dynamic PIDT processing, the LLAPI can only return data defined in the PIDT or data view record you specified. If you want a different set of data from the record, you must specify a different PIDT or data view record or request dynamic PIDT processing. If you request dynamic PIDT processing, all the data from the record is returned.

The LLAPI stores record response data in a response buffer anchored to the PIDT in field PIDTBUFP. If you do not request dynamic PIDT processing, the response data for each row is left-justified and padded with blanks on the right to the maximum size specified by field PIDTMAXL. (Exception: If you use an application-specified date format, the length of date values will be the larger of PIDTMAXL or the length of your date format.) List entry and multiple response items are separated by the response separator character in PIDT field PIDTSEPC. The LLAPI does not append a trailing separator character. The number of responses for the field is specified in PIDT field PIDTCNFR. For example, a list (data collected by using the list processor program exit) that looks like this on your display:

```
STMT1___
________
STMT3___
STMT4___
________
________
STMT7___
```

appears to your application, if you do not request dynamic PIDT processing, as:

```
'STMT1   ,,STMT3   ,STMT4   ,,,STMT7   '
```

The maximum length for each item is 8 characters (PIDTMAXL=8), and the value of PIDTSEPC is ’,’.

When retrieving list items, the field PIDTMNCR is set to 1, and the field PIDTCNFR is set to the number of the highest list item. For example, if items 1 through 10 in the list are blank and items 11 and 12 contain values, PIDTCNFR is set to 12.

Visible description and direct-add data that is longer than the size of field PIDTVISD are truncated.

This transaction cannot retrieve SRC records.

Group Prefix Processing Considerations

Record entries that have multiple p-words associated with a particular data item are called *group items*. PIDT rows corresponding to a group item have the PIDTGRPX field set to Y. These entries have their p-words stored in the PIPT table corresponding to the PIDT. The address of the PIPT table is stored in the PIDTPIPT field. The PIDTFPAT field holds the row offset in the PIPT table where the first p-word is stored. The PIPTFLAG for this entry contains X'40' to indicate the beginning of the group. The PIPT row entries are read until an entry is found that contains X'60' to indicate the end of the group. The p-words are stored in the PIPTPRFX field of each PIPT row. Refer to the *Tivoli Information Management for z/OS Panel Modification Facility Guide* for information on p-words.

Dynamic Record Retrieval Considerations

Your application can request the LLAPI to dynamically build a PIDT based on the data in the record. This means you can retrieve a record without first defining the view of the record in a PIDT.

Note: Dynamic record retrieval is not supported if a data view name is supplied with this transaction. Dynamic record retrieval is supported with bypass panel processing. However, the dynamic PIDT cannot be used in either a create or an update transaction. All references to PIDTs in this section discuss dynamically built PIDTs unless stated otherwise.

Ensure that you check out the record (T104) before you retrieve the record (T100) to prevent an update of the record by other users or applications.

To dynamically build a PIDT, the LLAPI uses another PIDT, which you specify, as a *model*. When the model PIDT is a create, update, inquiry, add relations, or retrieve PIDT, or a PIDT built by the BLGUT8 utility especially for use as a model for a dynamic PIDT, only the header row of the model PIDT is used in building the dynamic PIDT.

You can reuse a PIDT from a prior retrieve transaction to dynamically retrieve another record. When a PIDT is reused, the LLAPI determines whether it is big enough to accommodate the record being retrieved. If it is big enough, the PIDT is reused. If it is not big enough, the header is used as a model for generating another PIDT. The API automatically deletes the PIDT that was too small.

When selecting a non-dynamic PIDT to use as a model, consider the subsequent transactions, such as retrieve (T100), update (T105), or create (T102), in which you plan to use the PIDT. You might want to specify a model PIDT that has the highest authorization that subsequent transactions need.

To request a PIDT, your application must:
- Set PICADYNM to **Y**.
- Specify a PIDT in PICATABN.

Your application might also need to set PICAPIDT, depending on the type of PIDT specified as a model. Specify values for PICATABN and PICAPIDT in one of the following combinations:

- If PICATABN is a create, update, inquiry, add relations, or retrieve PIDT that is already in storage, PICAPIDT must be the address of the PIDT named in PICATABN. Save these values (PICATABN and PICAPITDT) so that you can free the model PIDT using T006. Your application must save the values and free the model because the LLAPI replaces the model's address in PICAPIDT with the address of the new dynamic PIDT but does not delete the model from storage.
- If PICATABN is a create, update, inquiry, add relations, or retrieve PIDT not in storage, PICAPIDT must be set to zeroes.
- If PICATABN is a header-only PIDT, PICAPIDT must be set to zeroes. (To build this type of PIDT run BLGUT8 with the word HEADER specified in the USE field.)
- If PICATABN is a PIDT from a previous retrieve transaction in this API session, PICAPIDT must be set to the address of the PIDT. (A dynamic PIDT is identified by a D in the PIDTUSEF field.)

  Reusing a PIDT causes the data in the entry rows to be overwritten if the PIDT is big enough to hold the record you are retrieving. If the reused PIDT is not big enough, it is deleted from storage after using the header rows as a model for the new PIDT. Get the address of the new PIDT after the retrieve is complete.

You can also set the following PICA fields for the PIDT that is to be dynamically built:

- PICAREQL can be set to add a number of bytes to the end of the returned response buffer. This additional storage in the response buffer can then be used to enlarge fields on an update transaction of the same record or a create transaction for a new record.
- PICAESPC can be set to a number of bytes to be added to the end of each data item in the response buffer. Each member of a list item or multiple response item is also followed by the number of bytes specified. This additional storage in the response buffer can then be used to enlarge fields on an update transaction of the same record or a create transaction for a new record.

When a retrieve transaction requests dynamic PIDT processing, the LLAPI builds PIDT entries for each Structured Description Entry (SDE) in the Tivoli Information Management for z/OS record with the exception of list data items. List data items have a single PIDT entry for each unique list. For nonreplaceable SDE items, PIDTs have one PIDT entry for each SDE item. This PIDT is used for updating the same record ID or creating new records. Therefore, no obtain resources transaction (T101 or T103) is required when you use a dynamic PIDT for a create (T102) or update (T105) transaction.

When the retrieve transaction returns, the following fields are set for dynamic PIDT processing:

- PIDTNAME contains the name of the PIDT specified in PICATABN with an asterisk (*) at the end. If a dynamic PIDT was specified as a model, the name is the same as the model's, including the asterisk (*).
- PIDTSPCP points to the beginning of the free space requested by PICAREQL.
- PIDTSPCE points to the end of the free space requested by PICAREQL.
- PIDTMAXL contains the length of the data retrieved for the PIDT entry plus the PICAESPC value specified. This value is based on the length of the data in the record and does not necessarily correspond to the actual maximum length for the field defined in your panels.
- PIDTRDEF is set to the letter O if the corresponding SDE in the Tivoli Information Management for z/OS record does not contain a p-word or an s-word.
- PIDTPIPT points to a PIPT or contains zeroes. The PIPT returned contains only prefix entries for SDEs that have more than one p-word associated with them. For list processor data, the LLAPI uses only the first entry of a unique list to obtain the prefix or prefixes stored with the list data. The PIPT has the same name as the dynamic PIDT.
- PIDTFPAT points to the row in the PIPT table where the first multiple p-word is stored.
- PIDTRTYP is set to **Y** for the first PIDT entry that has a matching s-word in the create control panel BLG1AACP. (BLG1AACP is searched.)
- PIDTDIAG is set to B if the corresponding SDE in the Tivoli Information Management for z/OS record marks the beginning of a dialog or to E if the corresponding SDE in the Tivoli Information Management for z/OS record marks the end of a dialog.

You can retrieve a record on a database then use the record ID of that record to create a record on a second database. The record ID of the retrieved record cannot match an existing record ID on the second database.

An all-numeric value of the record ID of the retrieved record might be greater than the next system-assigned record ID on the second database. In this case, Tivoli Information Management for z/OS uses the value of the record ID of the retrieved record plus one for the next system-assigned record ID. For example, if the retrieved record has a record ID of X'00002000' and the next system-assigned record ID when the retrieve transaction starts is X'00001000', the next system-assigned record ID after the retrieve transaction finishes will be X'00002001'. If the retrieved record has a record ID of X'00000900' and the next system-assigned record ID when the retrieve transaction starts is X'00001000', the next system-assigned record ID after the retrieve transaction finishes will be X'00001000'.

Ensure that you check out the record (T104) before you retrieve the record (T100) to prevent an update of the record by other users or applications.

Text Processing Considerations

When text entries are built, PIDTSYMB is assigned a value of `Xnnnn`, with `nnnn` being the number of the text entry in the record. For example, for the first text entry found in the record, PIDTSYMB is assigned the value X0001; for the next text entry found in the record, PIDTSYMB is assigned the value X0002. This value is also used instead of TEXT_TYPE_S-WORD_INDEX as the second qualifier of the data set name if data set processing was indicated. See the description of the text data set on page 21 for more information.

Multiple or List Data Item Processing Considerations

The first entry in a list determines the settings for the PIDT flags for all the other members of the list. The first member of a multiple response group determines the settings for the PIDT flags for all the other members of the multiple response group.

Group Prefix Processing Considerations

If a record entry is a group item, all multiple response entries and list items associated with the record entry belong to the first response of the entry. This means that all the p-words in the PIPT for a PIDT entry are prefixed to each multiple response in the data buffer for that PIDT entry if this PIDT is used for create or update transactions.

A dynamic PIPT is created for processing group items in the dynamic PIDT. This PIPT cannot be used for validation. The LLAPI uses only the first entry for a unique list to determine the p-words associated with that list.

Text Processing Considerations

To suppress text processing, set PICASTXT to Y. Any other value enables text processing. If you choose text processing, you can specify whether the application should return the text audit data. To suppress text audit data when text processing is enabled, set PICASAUD to Y. The default is to return text audit data. Your application can transfer text using an internal storage buffer or external data sets. The buffer provides faster data transfer, but it consumes storage. Data sets use less storage, but using them increases transfer time. Setting PICATXTP to B enables buffer transfer. Setting PICATXTP to D enables data set transfer. Data set transfer is the default method.

Text Buffer Transfer

PIDTCNFR is the number of text units (lines) transferred. Each text unit consists of text whose length your application specifies in PICATXTW. If audit data is not suppressed, each text unit will also contain an additional 36 bytes of audit data. For a description of the format of the audit data, see page 22. PIDTCURL contains the length of all text units

returned; if audit data is not suppressed, the length of the audit data is also included in this total. If PIDTCURL is divided by PIDTCNFR, the dividend should equal the text unit (line) width with no remainder.

Here is an example of how the buffer transfer function works. You want your API application to retrieve text and audit data from a Tivoli Information Management for z/OS database record and from a problem record. Assume you want to use a retrieve transaction to retrieve text from a problem record. If you look at the record interactively in Tivoli Information Management for z/OS, the text might look like this:

```
Bill called to say he could not log on to the
system this morning.

I asked Jim to resolve the problem.

Jim called back and said that Bill should be able to
log on now.

At 9:40 I called Bill back and had him try again.  He
was able to log on.
```

Assume that date, time, and user ID audit data was collected with the text.

Your API application sets the following fields in the PICA.

```
PICASTXT=N,PICATXTP=B,PICATXTU=4,PICATXTW=15,PICATXTA=B,PICASAUD=N
```

All of these fields deal with the use of the buffer transfer.

- PICASTXT=N (you want the text retrieved)
- PICATXTP=B (you want buffer processing)
- PICATXTU=4 (you want 4 text lines)
- PICATXTW=15 (you want a width of 15 characters of those lines)
- PICATXTA=B (you want the data from the bottom of the block of text).
- PICASAUD=N (you want the text audit data retrieved).

When your application runs under these conditions, the text you retrieve is an unbroken string of characters that looks like this:

```
'log on now.   91119 09:07:30 userid                          911
17 09:07:30 userid              At 9:40 I calle 91117
09:07:30 userid            was able to log 91117 09:07:30 userid
      '
```

The text string in the buffer is the first 15 characters of the last 4 lines in the bottom part of the original block of data, with each line followed by 36 bytes of audit data (`91117 09:07:30 userid` and padding blanks).

Data Set Transfer

PIDTCNFR is equal to 1. PIDTCURL contains the length of the data set name. The LLAPI stores each text type in the record in a separate sequential data set. PIDTDATP points to the name of the data set stored in the response buffer. See page 21 for more information about text data sets.

History Data Processing

You can obtain history data by setting PICAHIST to **Y**. The data is returned in the PIHT table, and its address is placed in PIDTPIHT. The PIHT consists of a header portion and a series of rows, where each row describes a piece of data. A history entry is composed of one

or more rows grouped in sequence. Entries created by Tivoli Information Management for z/OS Version 1 (PIHTVER1=**Y**) have only one row of data per group. All other entries (PIHTVER1≠**Y**) can have several rows forming a group, where the first row of each group has PIHTSGRP set to **Y**. When multiple rows are present, those with control data (PIHTCNTL=**Y**) appear before those with regular data (PIHTCNTL≠**Y**). Control data is journalized with FIRST specified. Regular history data is journalized with ORDER specified.

The PIHT is freed on a retrieve transaction if any of the following occur:

- The PIDT points to a PIHT that is not large enough to hold the history data associated with the retrieved record.
- No history data is available for the retrieved record and the PIDT contains an address of a PIHT.
- The retrieve transaction did not request history data (PICAHIST≠Y) and the PIDT contains an address of a PIHT.

Date Considerations

When PICADFMT=0 (that is, X'00'), any value specified in PICADSEP is ignored; that is, dates received by your application from the API will be in the same format as they are in the SDDS portion of the database.

If you want to receive dates in a different format, specify that format in the PICADFMT and PICADSEP fields (PICADFMT is described in 111 and PICADSEP is described in 111). If you choose this option and your date format is longer than PIDTMAXL for a field, the entire date will be returned and PIDTCURL will be larger than PIDTMAXL.

Field Specifications

You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T100.
PICATABN Record retrieval PIDT name. If you are using data models (**PICADMRC=Y**), then this is the data view name.
PICARNID External record ID or root VSAM key of record to retrieve.
PICASTXT Y if no text retrieval is required.

The following fields are optional:

PICATXTP Indicates buffer (B) or data set (D) processing.
PICATXTU Maximum number of text units (lines). Used only if PICATXTP=B.
PICATXTW Maximum text unit (line) width. Used only if PICATXTP=B.
PICATXTA Indicates bottom (B) or top (T) block of text returned. Used only if PICATXTP=B.
PICAHIST Y indicates history data processing.
PICAVSAM Y indicates value in PICARNID is the root VSAM key.
PICADYNM Y indicates dynamic record retrieval is requested.
PICAPIDT Address of already loaded PIDT
PICAREQL Number of bytes to add to the end of the response buffer. This is ignored if PICADYNM≠**Y**.
PICAESPC Number of bytes to add to the end of each response in the response buffer. This is ignored if PICADYNM≠**Y**.
PICAUSRN The name by which Tivoli Information Management for z/OS recognizes your application.You can specify a value here to change the name of the current application ID. Tivoli Information Management for z/OS uses this

name in place of a TSO user ID when performing record access privilege class processing. The value specified must be an eligible user of the current privilege class record.

PICACLSN A valid privilege class name.You can specify a value here to change the current privilege class record.

PICADFMT The index of the date format to use for exchanging datevalues between the API and your application.

PICADSEP The character slash (/) or hyphen (-) or period (.) used to separate month, day, and year portions of dates used in date formats which use a separator character.

Table 22 shows the retrieve record transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 22. LLAPI Transaction T100. Retrieve Record (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T100 (Retrieve Record) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICATABN=record retrieval PIDT name. If you are using data models (**PICADMRC=Y**), then this is the data view name. • PICARNID=record ID or root VSAM key of record to retrieve • PICAVSAM=Y if using a root VSAM key • PICASTXT=Y (if no text is wanted) • Other PICA text fields if you want text processing • Other PICA fields if you want dynamic PIDT retrieval • PICAHIST=Y if you want history data retrieval. • PICAUSRN=application ID if you want to change the name of the current application. • PICACLSN=privilege class if you want to change the privilege class. • PICASAUD=Y if you want to suppress text audit data from being returned with text data. ■ Calls BLGYSRVR(*PICA*).

Table 22. LLAPI Transaction T100 (continued). Retrieve Record (Synchronous)

Step	Program	Action
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following fields: • PICARETC • PICAREAS • PICAPIDT • PIDTBUFP • PIDTBUFL • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAPIDT points to PIDT. • PIDTBUFP points to response buffer. • PIDTBUFL contains length of response buffer. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Create Record (T102)

CAUTION:
You can damage your existing database if you do not use this transaction correctly.

This transaction creates records by processing PIDT entries. It extracts control data from the PIDT entries and response data from the response buffer, the PIDT, the PIPT, or all three. Leading and trailing blanks are removed from all but text data. Do not imbed blanks in a response or include the value in PIDTSEPC as part of a response. The LLAPI then puts the data in storage in the Tivoli Information Management for z/OS internal format. Record file processing completes the record creation in the database, and returns the record ID in PICARNID.

To assign a record ID to the record, you must supply data for the PIDT entry that collects the RNID/ prefix. For example, the problem record create PIDT uses S0CCF to define the user assigned record ID. You can define a record ID or you could have obtained a record ID by starting the T003 transaction. If you do not assign a record ID, the system assigns the next available record ID automatically. If you create a record using a dynamic PIDT built by a retrieve transaction, you can use the record ID contained in that PIDT. The LLAPI verifies that this record ID has not been used. If the record ID is all numeric and greater than the current last-used system assigned record ID, the last-used system assigned record ID is changed to the record ID from the PIDT. In that case, the next system assigned record ID is one larger than the record ID from the PIDT.

If you use panel processing, the LLAPI uses TSP BLGAPI02, user exits, and panel BLG1AACP to create records. The TSP uses some of the panels and selections that are used by interactive create processing, including the selection that runs program exit BLG01050 and the selection from the record summary panel that files the record. If you plan to create

records of your own type (including Tivoli Information Management for z/OS Integration Facility), have tailored your panels, or want to use existing panel automation see “Tailoring the Application Program Interfaces” on page 289, and “Terminal Simulator Panels” on page 349 for information on interface tailoring and LLAPI create processing. If you use bypass panel processing, the LLAPI uses TSP BLGAPIPX and user exits to create records.The TSP does not use your panels and selections.

You can use one of the three types of PIDTs. You cannot use dynamic PIDTs with data model records or with bypass panel processing. See “Dynamic Record Retrieval Considerations” on page 56 for more information about dynamic record retrieval.

The following steps outline the create record process:

The PIDT or data view record that you choose determines the record type and the fields that can be processed for the record. The name that you specify in PICATABN must match the name in PIDTNAME.

Are you using a dynamic PIDT (obtained by a record retrieve transaction (T100))?

Note: Dynamic is not allowed if you are using data model records or are bypassing panel processing.

- No, perform the following steps:

  1. Specify the name of the PIDT in PICATABN and, if you previously performed a T101, its address in PICAPIDT, or the name of the data view record in PICATABN.
  2. Specify your estimate of the total length of field values, including extra bytes for fields using separator characters, in PICAREQL.

     If the purpose of your application is to create multiple records of the same type, then allocate enough response buffer storage to satisfy the largest need.
  3. Decide your text processing medium (internal storage buffer or external data set).

     Note: You cannot mix storage buffer processing and data set processing. You must use one or the other.
  4. If you have not already obtained the create resources, obtain resources needed to create the record (T101).

     At T101 completion, PICAPIDT points to the PIDT, PIDTBUFP points to the response buffer, and PIDTBUFL contains the buffer length.
  5. Store the field-related values in the response buffer and set PIDT fields.

     For response and text fields, your application sets PIDTDATP to point to field data, PIDTCURL to field length, and PIDTCNFR to the number of responses in the buffer.

     For phrase (PIDTRDEF=P) and direct add (PIDTRDEF=D) type items, your application stores no data in the response buffer because the data is already contained in PIDTVISL. Your application sets PIDTCNFR to 1 in order to cause the LLAPI to store the phrase or direct add type items in the record.
  6. Perform record creation (T102).
  7. Free data table (T006).

If your application creates multiple records of the same type, with the same data view (PIDT), defer this step until you create the last record.

8. Free any text data sets created unless you want to reuse them.
9. Other programming techniques to consider when creating multiple records of the same type are:
   - Fixed field lengths (for example, Hardware Component Generic device type):
     a. Save the buffer locations of the data on the first pass.
     b. Overwrite the data in the buffer.
   - Varying field length:
     a. Allocate the response buffer using the sum of the values stored in PIDTMAXL as the response buffer size. This allows space in the response buffer for the maximum length of each field.
     b. For each record create, overwrite the data in the response buffer.
     c. Update field PIDTCURL.
   - If you use different fields, reinitialize the used PIDT rows, free (T006) the record create resources, and reallocate them (T101) before storing data for the next record.

■ Yes, perform the following steps:

1. Specify its name in PICATABN.
2. Specify its address in the PICAPIDT field, and set the PICADYNM field to **Y**.
3. Decide your text processing medium (internal storage buffer or external data set).

   Note: You cannot mix storage buffer processing and data set processing. You must use one or the other.

   The record retrieve transaction (T100) returns the response buffer and sets the PIDTBUFP field.
4. Store the field-related values in the response buffer and set PIDT fields.

   For response and text fields, your application sets PIDTDATP to point to field data, PIDTCURL to field length, and PIDTCNFR to the number of responses in the buffer.

   For phrase (PIDTRDEF=P), direct add (PIDTRDEF=D), and other type items (PIDTRDEF=O), your application stores no data in the response buffer because the data is already contained in PIDTVISL. Your application sets PIDTCNFR to 1 in order to cause the LLAPI to store the phrase, direct add, or other type items in the record.

   Optionally set PIDTDELO to **Y** to indicate that all PIDTRDEF=O entries are to be excluded from the record.
5. Perform record creation (T102).
6. Free data table (T006).

   If your application creates multiple records of the same type, with the same data view (PIDT), defer this step until you create the last record.
7. Free any text data sets created unless you want to reuse them.

Multiple or List Data Item Processing Considerations

When you collect multiple or list-item responses, the responses must be separated by the separator character specified in PIDT field PIDTSEPC. Responses do not require padding with blanks. Do not append a separator character to the last response of a field.

In a dynamic PIDT where you specified a value for PICAESPC, each member of a list item that was retrieved is followed by the number of blanks specified in PICAESPC.

An example of a list item using a comma separator character is `moda,modb,modc`. An example of a skipped entry is `moda,,modc`. (the first entry contains `moda` and the third entry contains `modc`). A null or blank list entry causes the API to skip the list item in that position.

The API does not support multiple response list items.

Text Audit Data Considerations

You can include audit data with freeform text or allow the LLAPI to determine the applicable audit data for freeform text. Set PICATXAU to **Y** to indicate that each line of input text contains a fixed audit data structure at the end of the line. The audit data structure must be of the format described on page 22. A blank occurs between each field, and three blanks follow the final audit data field.

Each audit data field (date, time, application or user ID, and privilege class) is assessed separately. A blank in the first position of an audit data field means the field is empty. Data in an audit field is delimited by the first blank found in the field or by the end of the field. If you set PICATXAU to Y, each line of input text must have data in at least one of the audit data fields.

When PICATXAU is set to **Y**, TSP BLGAPI02 sets a nonzero PICAREAS code if text audit data processing is disabled. If enabled, your application must be running under a privilege class that allows database administrator authority to set PICAHIST to **Y**. TSP BLGAPI02 can be modified to allow applications to set PICATXAU to **Y** to enable text audit data processing and to change the level of authority required to set PICATXAU to **Y**.

Dynamic PIDT Considerations

You can use a dynamic PIDT by performing the following steps:

- PICADYNM to **Y**
- Specifying its name in PICATABN
- Ensuring that PICAPIDT contains the address of the dynamic PIDT.

Note: Dynamic PIDT processing is not supported if a data view name is supplied with this transaction or if bypass panel processing was specified at initialization.

The dynamic PIDT must be requested in a retrieve record transaction. No obtain record create resource transaction is required.

When PICADYNM is set to Y, TSP BLGAPI02 sets a nonzero PICAREAS code if the dynamic PIDT processing function is disabled. If enabled, your application must be running under a privilege class that allows database administrator authority to set PICADYNM to **Y**. TSP BLGAPI02 can be modified to allow applications to set PICADYNM to **Y** to enable the dynamic PIDT processing function and to change the level of authority required to set PICADYNM to Y.

When a record is retrieved, the dynamic PIDT has one entry for every SDE in the record with the exception of list data items. List data items have a single PIDT entry for each unique list. With nonreplaceable SDE items, dynamic PIDTs have one PIDT entry for each SDE item.

If a PIDT entry for a record access panel is passed on an update transaction with PIDTCHNG set to **Y**, that PIDT entry is added to the record instead of the PIDT entry normally added at file time. The record access panel PIDT entry uses the s-word associated with s-word index S0E17, and it must be a direct add (D) type PIDT entry.

You can alter data in the PIDT by changing the appropriate fields. For example, no existing other type entries (PIDTRDEF=O) are included if the application sets the PIDTDELO field to Y. If the PIDTDELO is something other than **Y**, the LLAPI processes the other type entries on a one-by-one basis.

If the record is checked out when your application uses the dynamic retrieval transaction, the PIDT contains an entry for the name of your application. If you then use this PIDT for a create transaction, you might want to exclude that entry. If you keep it in the PIDT, the newly created record is checked out to your application ID.

For a dynamic PIDT, the application can set the following additional PIDT fields. Incorrect modification of these fields can cause damaged records to be stored in the database.

Note: Do not set these fields in a PIDT that is not dynamic.

PIDTSYMB Visible form of the s-word index, or the character string Xnnnn, if retrieving freeform text with a dynamic PIDT where nnnn starts at 0001 and increases with each freeform text item in the unique text record. This field contains a symbolic name for a dynamic PIDT only if an s-word index is present or this is a text entry.

PIDTDATE Field defined as a date. Any p-word beginning with DAT is automatically considered to be a date.

PIDTMAXL Maximum length of a PIDT entry's data. If this increases, the data in the buffer for this entry can be moved to the free space area of the buffer. See "Increasing the Length of a Field" on page 68 for information on how to do this.

PIDTMNCR Maximum number of responses for a PIDT entry. If this increases, the data in the buffer can also be moved to the free space area of the buffer.

PIDTSRCH Field defined as searchable.

PIDTJRNL Field defined as journalized.

PIDTPNLN Panel name entry. If you change the panel name, be sure the panel type remains the same.

PIDTINDX Internal s-word or p-word index. For dynamic PIDT entries of type other (PIDTRDEF=O), this field contains the response number of the panel.

- If you change this when it contains an s-word index, change PIDTSWDD and PIDTSYMB to correspond.

- If you change this when it contains a p-word index, change PIDTPFXD and PIDTSYMB to correspond.
- If you change this when it contains a response number, you need not change a corresponding field.
- If you change a response number to an s-word or a p-word index, be sure to set all the necessary fields.

PIDTSWDD S-Word. If you change this, change PIDTINDX to correspond. If you change the length, change PIDTSWDL to correspond.

PIDTPFXD P-Word. If you change the length, change PIDTPFXL to correspond.

PIDTSWDL Length of s-word field, PIDTSWDD.

PIDTPFXL Length of p-word field, PIDTPFXD.

PIDTREPL Field defined as replaceable.

PIDTRTYP Field defined as record type. If you change this, be sure the s-word for the entry you set to define the record type is defined in the BLG1AACP panel.

PIDTDIAG Field identifies a dialog begin (B) or dialog end (E).

You can retrieve a record on a database then use the record ID of that record to create a record on a second database. The record ID of the retrieved record cannot match an existing record ID on the second database.

An all-numeric value of the record ID of the retrieved record might be greater than the next system-assigned record ID on the second database. In this case, Tivoli Information Management for z/OS uses the value of the record ID of the retrieved record plus one for the next system-assigned record ID. For example, if the retrieved record has a record ID of X'00002000' and the next system-assigned record ID when the retrieve transaction starts is X'00001000', the next system-assigned record ID after the retrieve transaction finishes will be X'00002001'. If the retrieved record has a record ID of X'00000900' and the next system-assigned record ID when the retrieve transaction starts is X'00001000', the next system-assigned record ID after the retrieve transaction finishes will be X'00001000'.

Increasing the Length of a Field

If PICAREQL was set in the record retrieve transaction, the PIDTSPCP field points to the beginning of the free space in the response buffer that was allocated with the retrieve record transaction. The PIDTSPCE field points to the end of that same free space. One method for increasing a field's length beyond the value in PIDTMAXL, or for increasing the response count beyond the value in PIDTMNCR, is the following:

- Check whether the new length plus PIDTSPCP is less than or equal to the value in PIDTSPCE. If it is not, the buffer is not large enough. The application must retrieve the record again and request a larger buffer.
- If the buffer is large enough:
  1. Set PIDTDATP to the value in PIDTSPCP.
  2. Set PIDTSPCP equal to PIDTSPCP plus the new field length.
  3. Change PIDTMAXL and PIDTCURL to the new field length.
  4. Write data in the buffer area pointed to by PIDTDATP.

The application cannot create a record if an appropriate summary panel for the record type is not defined in the create control panel BLG1AACP.

If your application changes record entries defined by the following s-word, the LLAPI ignores these PIDT:
XIM00SST00 The timestamp for when the record was created.
XIM00SST01 The timestamp for when the record was updated.

Group Prefix Processing Considerations

Record entries that have multiple p-words associated with a particular data item are called group items. PIDT rows corresponding to a group item have the PIDTGRPX field set to Y. These entries have their p-words stored in the PIPT table corresponding to the PIDT. The address of the PIPT table is stored in the PIDTPIPT field. The PIDTFPAT field holds the row offset in the PIPT table where the first p-word is stored. The PIPTFLAG for this entry contains X'40' to indicate the beginning of the group. The PIPT row entries are read until an entry is found that contains X'60' to indicate the end of the group. The p-words are stored in the PIPTPRFX field of each PIPT row.

For dynamic PIDTs, you cannot use a PIPT created for group items for validation.

History Data Considerations

You can include history data by setting PICAHIST to **Y** and ensuring that PIDTPIHT contains the address of your history data. The data must be requested as history data on the retrieve record transaction (T100). If you use a PIDT obtained by the record create resource transaction instead of a dynamic PIDT, the PIHT address must be copied into the new PIDTPIHT from the one returned by record retrieve.

You can modify the history data by changing the appropriate fields in the PIHT. For example, the data has a maximum length (PIHTMAXL) and a current data length (PIDTCURL). The current length can be increased up to the maximum. The application can delete a PIHT row by setting the current data length field to the value of zero. When deleting the first row in a group, ensure that the start history group flag (PIHTSGRP) is set to **Y** for the new first row.

When a record is created in the database, normal processing adds real-time history entries for any fields that have the journal flag turned on. These history entries are in addition to those added by setting PICAHIST to Y. The application can control the creation of the real-time entries by setting the corresponding journal flags appropriately. If you use bypass panel processing, the LLAPI uses TSP BLGAPIPX and user exits to create records. The TSP does not use your panels and selections.

When PICAHIST is set to **Y**, TSP BLGAPI02 or BLGAPIPX sets a nonzero PICAREAS code if the history data processing function is disabled. If enabled, your application must be running under a privilege class that allows database administrator authority to set PICAHIST to **Y**. TSP BLGAPI02 or BLGAPIPX can be modified to allow applications to set PICAHIST to **Y** to enable the history data processing function and to change the level of authority required to set PICAHIST to Y.

Date Considerations

When PICADFMT=0 (that is, X'00'), then any value specified in PICADSEP is ignored; that is, dates passed by your application to the API will be in the same format as they are in the SDDS portion of the database.

If you want to pass dates in a different format, specify that format in the PICADFMT and PICADSEP fields (PICADFMT is described on page 111 and PICADSEP is described on page 111). The API will convert the dates you pass into the default external date format specified in the session parameters before they are processed by Tivoli Information Management for z/OS. If you choose this option and your date format is longer than PIDTMAXL for a field, then set PIDTCURL to the length of your date. You will not receive a length error unless the date is longer than PIDTMAXL after it has been converted to default external date format.

Field Specifications

You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T102.

PICAPIDT Pointer to the PIDT.

If you are using equal sign processing, you will need to specify:

PICAEQRP Set to **Y**. If the response data (or visible phrase for direct-add items) contains an equal sign (=) then the data will be processed as equal data and processed according to the rules defined by the product.

When you are not using a dynamic PIDT, you must specify the following PIDT fields when processing responses:

PIDTDATP Pointer to data location in the response buffer (except for visible phrase and direct add items).

PIDTCURL Length of response or responses in the response buffer (except for visible phrase and direct add items).

PIDTCNFR Current number of responses for the field.

When you are using a dynamic PIDT, you can specify the following PIDT fields when processing responses:

PIDTDELO Set to **Y** for all other type (PIDTRDEF=O) entries so that they are not included in the record.

PIDTCURL Length of responses in the response buffer if they have been changed (except for visible phrase and direct add items).

PIDTCNFR If changed, current number of responses for the field. Set to zero to exclude a field.

PIDTMNCR If increased, maximum number of responses for the field.

PIDTMAXL If increased, maximum length of responses for the field.

PIDTDATP If PIDTMAXL or PIDTMNCR was increased and you moved the data to the free space area of the buffer.

PICAUSRN The name by which Tivoli Information Management for z/OS recognizes your application. You can specify a value here to change the name of the current application ID. Tivoli Information Management for z/OS uses this name in place of a TSO user ID when performing record access privilege class processing. The value specified must be an eligible user of the current privilege class record.

PICACLSN A valid privilege class name. You can specify a value here to change the current privilege class record.

PICADFMT The index of the date format to use for exchanging date values between the API and your application.

PICADSEP The character slash (/) or hyphen (-) or period (.) used to separate month, day, and year portions of dates used in date formats which use a separator character.

If you choose buffer processing for text:
- PICATXTP = B
- PICATXAU = **Y** if text audit data is specified
- PIDTDATP = pointer to text in the response buffer
- PIDTCURL = total text length
- PIDTCNFR = number of text units (lines) being processed

If you choose data set processing for text:
- PICATXTP = D
- PICATXAU = **Y** if text audit data is specified
- PIDTDATP = pointer to data set name in the response buffer
- PIDTCURL = length of data set name
- PIDTCNFR = 1

If you choose history data processing:
- PICAHIST = **Y**
- PIDTPIHT = pointer to PIHT
- PIHTCURL > 0 for each data-entry to be added

If you choose equal sign processing:
- PICAEQRP = **Y**
- PIDTDATP = pointer to equal sign (=) in response buffer

You must specify the following field when processing a dynamic PIDT:
- PICADYNM=**Y** for dynamic PIDTs

Table 23 on page 72 shows the create record transaction flow for a synchronous environment. It is assumed that create record resources are obtained. For more detailed information on the LLAPI structures and their fields, see “LLAPI Structures” on page 100.

Table 23. LLAPI Transaction T102. Create Record (Synchronous)

Step	Program	Action
1	Application	■ Stores response data for each PIDT field (except visible phrase and direct add items) at the address in the response buffer. Each response can be no longer than the PIDTMAXL value and the number of responses cannot exceed the value in PIDTMNCR (except for list item fields). ■ When using a PIDT from the obtain record create resource transaction (T101): • Sets PIDTDATP. • Sets PIDTCURL to the data length. ■ When using a dynamic PIDT built by the retrieve record transaction (T100): • Sets PIDTDATP address if it is moved from the original location. • Sets PIDTCURL if the length increases or decreases from what it was initially. • Can alter PIDTMAXL and PIDTMNCR. Sets PIDTCNFR to the current number of field responses for each field. ■ Sets PICA fields as follows: • PICATRAN=T102 (Create Record) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICADYNM=Y if using a dynamic PIDT • PICAPIDT=address of record create PIDT or address of dynamic PIDT. • PICAUSRN=application ID if you want to change the name of the current application. • PICACLSN=privilege class if you want to change the privilege class. ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICARNID • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICARNID contains record ID. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Update Record (T105)

CAUTION:
You can damage your existing database if you do not use this transaction correctly.

This transaction updates records by processing PIDT entries and data from the PIPT. It extracts control data from the PIDT entries and response data from the response buffer. Leading and trailing blanks are removed from all but text data. Do not imbed blanks in a response or include the value in PIDTSEPC as part of a response. The LLAPI then puts the data in storage in the Tivoli Information Management for z/OS internal format. Record file processing completes the record update in the database.

You can use one of the three types of PIDTs. You cannot use dynamic PIDTs with bypass panel processing or data model records. See "Dynamic Record Retrieval Considerations" on page 56 for more information about dynamic record retrieval.

Note: Use of a dynamic PIDT is not supported with either bypass panel processing or data model records. See "Dynamic Record Retrieval Considerations" on page 56 for more information. If you choose dynamic processing (PICADYNM=Y), the PIDT and its data must have been obtained by requesting a retrieve transaction (T100) with the same record ID as the record to be updated and in the same Tivoli Information Management for z/OS database, and the record must not have been updated since the record was retrieved with the retrieve transaction (T100).

If you use panel processing, the LLAPIuses TSP BLGAPI05 to perform the update transaction. TSP BLGAPI05 performs the update command on the specified record, and then flows to the regular update target panel. The name of the panel flowed to after completing the update command must match the name of the panel specified in panel BLG1AAUP. To use the panel specified in panel BLG1AAUP as the summary panel, specify an authorization code of 0001 for that panel in BLG1AAUP. With panel processing, to update records of your own record type, you must modify control panel BLG1AAUP. See "Tailoring the Application Program Interfaces" on page 289 and "Terminal Simulator Panels" on page 349 for information that can help you understand what changes are required to update records of your own type.

If you use bypass panel processing, the LLAPI uses TSP BLGAPIPX to perform the update processing. It uses user exits to perform the update.

Note: If you are using logical database partitioning, you can update a record only if the Owning Partition of that record matches the Primary Partition of your privilege class.

Checking out the record before the update ensures that no other users can update the record prior to your update. Your administrator can define a time limit for checked out records (in the BLX-SP parameter APICHKOUTLIM, described in the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*) so that records will not inadvertently remain indefinitely checked out if your application does not check in the record.

You can determine how you want to process lists on update. That is, you can simply update lists (this is the default), you can append new list items to existing lists, or you can replace existing lists. To specify the type of update, set PICALSTM to indicate whether you want to update, append, or replace list items.

The following steps outline the update record process:

When you are not using a dynamic PIDT, perform the following steps:

1. Specify the name of the PIDT in PICATABN and its address in PICAPIDT (if you previously perform a T103) or the name of the data view record in PICATABN.

   The PIDT can be a static PIDT or a PIDT generated from a data view record. The static PIDT or data view record you choose determines the record type and the fields that can be processed for the record. If you choose a PIDT defined for a record type different from the record type being updated, the transaction terminates with an error.

2. Specify your estimate of the total length of field values, including extra bytes for fields using separator characters, in PICAREQL.

   If the purpose of your application is to update multiple records of the same type, then allocate enough response buffer storage to satisfy the largest need.

3. Decide your text processing medium (buffer or data set).

   Note: You cannot mix buffer processing and data set processing. You must use one or the other.

4. Create text data sets if needed.

5. If you are not using a PIDT previously obtained using T103, obtain resources needed to update the record (T103).

   At T103 completion, PICAPIDT points to the PIDT, PIDTBUFP points to the response buffer, and PIDTBUFL contains the buffer length.

6. Store the field-related values in the response buffer and set PIDT fields.

   For response and text rows, your application sets PIDTDATP to point to field data, PIDTCURL to field length, and PIDTCNFR to the number of responses in the buffer.

   For phrase (PIDTRDEF=P) and direct add (PIDTRDEF=D) type items, your application stores no data in the response buffer as the data is already contained in PIDTVISL. Your application sets PIDTCNFR to 1 in order to cause the LLAPI to store the phrase or direct add type items in the record. To delete phrase or direct add type items, your application must set PIDTCURL and PIDTCNFR to 1 and PIDTDATP to point to a separator character in the response buffer.

7. Check out the record (T104), when required to maintain data integrity.

8. Perform T105 to perform a record update. If another application or user is attempting to update the record, the record might be unavailable. You can direct the LLAPI to either retry this transaction from 1 to 255 times before returning control to your application or wait until the record is available. See page 18 for more information.

9. Check in the record (T008), if it is checked out. If another application or user is attempting to update the record, the record might be unavailable. You can direct the LLAPI to either retry this transaction from 1 to 255 times before returning control to your application or wait until the record is available. See page 18 for more information.

10. Free data table (T103).

    If your application is updating multiple records of the same type, with the same data view (PIDT), defer this step until you update the last record.

11. Free any text data sets created unless you want to reuse them.

When you are using a dynamic PIDT (obtained by a retrieve transaction (T100)), perform the following steps:

1. Specify the name of the PIDT in PICATABN, specify its address in the PICAPIDT field, and set the PICADYNM field to **Y**. The name that you specify in PICATABN must match the name in PIDTNAME.
2. Decide your text processing medium (internal storage buffer or external data set).

   Note: You cannot mix storage buffer processing and data set processing. You must use one or the other.
3. Create text data sets if needed.
4. The record retrieve transaction (T100) returns the response buffer and sets the PIDTBUFP field.
5. The retrieve stores the field-related values in the response buffer and sets PIDT fields.

   For response and text rows, your application sets PIDTDATP to point to field data, PIDTCURL to field length, and PIDTCNFR to the number of responses in the buffer.

   For phrase (PIDTRDEF=P), direct add (PIDTRDEF=D), and other type items (PIDTRDEF=O), your application stores no data in the response buffer as the data is already contained in PIDTVISL. Your application sets PIDTCNFR to 1 in order to cause the LLAPI to store the phrase, direct add, or other type items in the record. To delete phrase, direct add, or other type items, your application must set PIDTCURL and PIDTCNFR to 1 and PIDTDATP to point to a separator character in the response buffer.

   The PIDTCHNG field must be set to **Y** for that PIDT entry to be processed. The only exception to this rule is when the PIDTDELO flag is set to **Y**, the interface deletes all the other type (PIDTRDEF=O) items regardless of how the PIDTCHNG flag is set.
6. Perform T105 to perform a record update. If another application or user is attempting to update the record, the record might be unavailable. You can direct the LLAPI to either retry this transaction from 1 to 255 times before returning control to your application or wait until the record is available. See page 18 for more information.
7. Check in the record (T008), if it is checked out. If another application or user is attempting to update the record, the record might be unavailable. You can direct the LLAPI to either retry this transaction from 1 to 255 times before returning control to your application or wait until the record is available. See page 18 for more information.
8. Free data table (T103).

   If your application is updating multiple records of the same type or retrieving and updating records dynamically, with the same data view (PIDT), defer this step until you update the last record.
9. Free any text data sets created unless you want to reuse them.

When you are not using a dynamic PIDT, other programming techniques to consider when updating multiple records of the same type are:

1. If the varying fields' lengths are fixed (for example, Hardware Component Type):
   a. Save the buffer locations of the data on the first pass.
   b. Overwrite the data in the buffer.
2. If the varying fields' lengths vary:

a. Allocate the response buffer using the sum of the values stored in PIDTMAXL. This allows space in the response buffer for the maximum length of each field.

b. For each record update, overwrite the data in the response buffer.

c. Update field PIDTCURL.

3. If you use different fields, reinitialize the used PIDT rows or free the record update resources and reallocate (T103) before storing data for the next record.

Multiple or List Data Item Processing Considerations

The following rules apply when you collect multiple or list-item responses:

- The responses must be separated by the separator character specified in PIDT field PIDTSEPC.
- Each response does not require padding blanks.
- Do not append a separator character to the last response of a field.

An example of a list item using a comma separator character is `moda,modb,modc`.

An example of a skipped entry is `moda, ,modc`. The first entry contains `moda` and the third entry contains `modc`. The blank between the commas is required to indicate that any existing second list entry is to be skipped. A response of `moda,,modc` indicates to delete the second entry.

In a dynamic PIDT where you specified a value for PICAESPC, each entry of a retrieved list is followed by the number of blanks specified in PICAESPC.

The API does not support multiple response list items.

The number of response items is indicated in field PIDTCNFR. To delete a list response, two consecutive separator characters are stored in the response buffer with the second separator character logically replacing the deleted response. A separator character in the first position of the list item data indicates that the first list position item is to be deleted. A trailing separator character (after the last item in the input buffer) indicates that the next list item of that type in the record is to be deleted.

You can choose to update existing lists (the default), append new data to existing lists, or replace existing lists. You use PICALSTM to specify how lists should be processed.

This example 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 Performed
Update           Segment              Update


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



ADD              'ADD,BUILD1,,COPY'   ADD           Delete third
```

```
BUILD1                                  BUILD1        item on list
DELITEM                                 --------
COPY                                    COPY
--------                                --------
CHECK                                   CHECK
INIT                                    INIT



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

This example shows an update transaction appending data to an existing list of routine names. For this transaction, the figure shows: the list before the transaction on the left, the response buffer segment used to append to the list, and the results of the append.

```
List Before       Response Buffer       List After    Action Performed
Append            Segment               Append


ADD               'BUILD0,BUILD1,,COPY' ADD           Append new
BUILD1                                  BUILD1        items to list
DELITEM                                 DELITEM
COPY                                    COPY
--------                                --------
CHECK                                   CHECK
INIT                                    INIT
                                        BUILD0
                                        BUILD1
                                        --------
                                        COPY
```

This example shows an update transaction replacing from an existing list of routine names. For this transaction, the figure shows: the list before the transaction on the left, the response buffer segment used to delete data from the list, and the results of the delete.

```
List Before       Response Buffer       List After    Action Performed
Delete            Segment               Delete


ADD               'BUILD0,,BUILD2'      BUILD0        New data replaces
BUILD1                                  --------      old data
DELITEM                                 BUILD2
COPY
--------
CHECK
INIT
```

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

Text Considerations

To update text in the record, your application stores the data set name or text in the response buffer in the same way the API stores it during a record retrieval. A text data set name can be up to 44 characters long. Text data can be appended to, or it can replace, text currently in the record. To replace existing text with new text, set PICATXTR to **Y**. Each type of text

specified replaces any existing text of that type. To append new text to existing text, set PICATXTR to **N**. Each type of text specified is appended to existing text of that type. Existing text is deleted by setting PICATXTR to **Y** and specifying a single separator character as the response for the text item. If TEXTAUD=YES in the session-parameters member (BLGPARMS), existing text cannot be deleted or replaced.

Text Audit Data Considerations

You can include audit data with freeform text or allow the LLAPI to determine the applicable audit data for input freeform text. Set PICATXAU to **Y** to indicate that each line of input text contains a fixed audit data structure at the end of the line. The audit data structure must be of the format described on page 22. A blank occurs between each field, and three blanks follow the final audit data field.

Each audit data field (date, time, application or user ID, and privilege class) is examined separately. A blank in the first position of an audit data field means the field is empty. Data in an audit field is delimited by the first blank found in the field or by the end of the field as defined in the audit data structure on page 22. If you set PICATXAU to **Y**, each line of input text must have data in at least one of the audit data fields.

If PICATXAU is set to **Y** and text audit data processing is disabled in TSP BLGAPI05, TSP BLGAPI05 sets a nonzero PICAREAS code. If text audit data processing is enabled, your application must be running under a privilege class that allows database administrator authority to set PICATXAU to **Y**. TSP BLGAPI05 can be modified to allow applications to set PICATXAU to **Y** to enable text audit data processing and to change the level of authority required to set PICATXAU to **Y**.

Dynamic PIDT Considerations

Note: Use of dynamic PIDTs is not supported if bypass panel processing was specified at initialization or if a data view name is supplied with this transaction.

To use a dynamic PIDT, set PICADYNM to **Y** and ensure that PICAPIDT contains the address of your dynamic PIDT. The dynamic PIDT must have been obtained by requesting a retrieve record transaction with the same record ID as the record to be updated and on the same database. The record must not have been updated between the time of the retrieve transaction (T100) and the time of the update transaction (T105). The obtain record update resource transaction is not required because the retrieve transaction (T100) obtains all resources required by the LLAPI.

When PICADYNM is set to Y, TSP BLGAPI05 sets a nonzero PICAREAS code if the dynamic-PIDT processing function is left disabled. If enabled, your application must be running under a privilege class that allows database administrator authority to set PICADYNM to Y. TSP BLGAPI05 can be modified to allow applications to set PICADYNM to **Y** to enable the dynamic PIDT processing function and to change the level of authority required to set PICADYNM to **Y**.

When a record is retrieved, the dynamic PIDT has one entry for every SDE in the record with the exception of list data items. List data items have a single PIDT entry for each unique list. With nonreplaceable SDE items, dynamic PIDTs have one PIDT entry for each SDE item.

If the PIDT entry for a record access panel is passed on an update transaction with PIDTCHNG set to **Y**, that PIDT entry is added to the record instead of the PIDT entry normally added at file time. The record access panel PIDT entry uses s-word S0E17, and it must be a direct add (D) type PIDT entry.

You can alter data in the PIDT by changing the appropriate fields. For example, all existing Other type entries (PIDTRDEF=O) can be deleted by setting the PIDTDELO field to **Y**. If the PIDTDELO is something other than **Y**, the LLAPI processes the Other type entries on a one-by-one basis. Only PIDT entries with the PIDTCHNG field set to **Y** are processed.

If you change the s-word for a list, your application must pass all of the list data. Otherwise, the update transaction does not replace the s-words for all entries in the list.Your application must never alter the contents of the PIDTVLDD or PIDTVLDL fields. Existing data must not be changed between the time the update command is issued in API TSP BLGAPI05 and the time BLGYAPBR is called in API TSP BLGAPI05. Failure to observe these restrictions can cause unpredictable results.

For a dynamic PIDT only, the application can set the following additional PIDT fields. Incorrect modification of these fields can cause damaged records to be stored in the database:

PIDTSYMB Visible form of the s-word index or the character string Xnnnn if retrieving freeform text with a dynamic PIDT where nnnn starts at 0001 and increases with each unique freeform text item in the unique record.

PIDTDATE Field defined as a date.

Any p-word beginning with DAT is considered a date.

PIDTMAXL Maximum length of a PIDT entry's data. If this is increased, the data in the buffer for this entry can be moved to the free space area of the buffer. See "Increasing the Length of a Field" on page 80 for information on how to do this.

PIDTMNCR Maximum number of responses for a PIDT entry. If this is increased, the data in the buffer can also be moved to the free space area of the buffer.

PIDTSRCH Field defined as searchable.

PIDTJRNL Field defined as journalized.

PIDTPNLN Panel name entry. If you change the panel name, be sure to keep it the same type.

PIDTINDX Internal s-word or p-word index. For dynamic PIDT entries of type Other (PIDTRDEF=O), this field contains the response number on the panel. If you change this field when it contains:

- An s-word index, change PIDTSWDD and PIDTSYMB to correspond
- A p-word index, change PIDTPFXD and PIDTSYMB to correspond
- A response number, you need not change a corresponding field.

If you change a response number to an s-word or a p-word index, be sure to set all the necessary fields.

PIDTSWDD S-Word. If you change PIDTSWDD, change PIDTINDX to correspond. If you change the length, change PIDTSWDL to correspond.

PIDTPFXD P-Word. If you change the length, change PIDTPFXL to correspond.

PIDTSWDL Length of s-word field, PIDTSWDD.

PIDTPFXL Length of p-word field, PIDTPFXD.

PIDTREPL Field defined as replaceable.

PIDTRTYP Field defined as record type. If you change this, be sure the entry you set to define the record type has its s-word in the BLG1AUCP panel.

PIDTDIAG Field identifies a dialog begin (B) or dialog end (E).

Increasing the Length of a Field

If PICAREQL was set in the record retrieve transaction, the PIDTSPCP field points to the beginning of the free space in the response buffer that was allocated with the retrieve record transaction. The PIDTSPCE field points to the end of that same free space. A method to increase a field's length beyond the value in PIDTMAXL, or to increase the response count beyond the value in PIDTMNCR, is to follow these steps:

- Check whether the new length plus PIDTSPCP is less than or equal to PIDTSPCE value. If it is not, the buffer is not large enough. The application must do another retrieve record transaction and request a larger buffer.
- If the buffer is large enough:
  1. Set PIDTDATP to the value in PIDTSPCP.
  2. Set PIDTSPCP equal to PIDTSPCP plus the new field length.
  3. Change PIDTMAXL and PIDTCURL to the new field length.
  4. Write data in the buffer area pointed to by PIDTDATP.

The application cannot update a record if an appropriate summary panel for the record type is not defined in the update control panel BLG1AAUP.

To use the panel specified in panel BLG1AAUP as the summary panel instead of the regular target panel of the update, specify an authorization code of `0001` for that panel in BLG1AAUP.

If your application changes record entries defined by the following s-words, the LLAPI ignores these PIDT entries:

XIM00SST00 The timestamp for when the record was created

XIM00SST01 The timestamp for when the record was updated.

Group Prefix Processing Considerations

Record entries that have multiple p-words associated with a particular data item are called *group items*. PIDT rows corresponding to a group item have the PIDTGRPX field set to Y. These entries have their p-words stored in the PIPT table corresponding to the PIDT. The address of the PIPT table is stored in the PIDTPIPT field. The PIDTFPAT field holds the row offset in the PIPT table where the first p-word is stored. The PIPTFLAG for this entry contains X'40' to indicate the beginning of the group. The PIPT row entries are read until an entry is found that contains X'60' to indicate the end of the group. The p-words are stored in the PIPTPRFX field of each PIPT row.

For a dynamic PIDT, a PIPT for group items cannot be used for validation.

History Data Considerations

You can include history data by setting PICAHIST to **Y** and ensuring that PIDTPIHT contains the address of your history data. The data is obtained by requesting history data on

a retrieve record transaction with the same record ID as the record to be updated and on the same database. If you are not using a dynamic PIDT, copy the PIHT address into the PIDTPIHT field in this non-dynamic PIDT.

You can modify the history data by changing the appropriate fields in the PIHT. For example, the data has a maximum length (PIHTMAXL) and a current data length (PIHTCURL). The current length can be increased up to the maximum. The application can delete a PIHT row by setting the current data length field to the value of zero. When deleting the first row in a group, ensure that the start history group flag (PIHTSGRP) is set to **Y** for the new first row.

When a record is updated on the database, normal processing adds real-time history entries for any fields that have the journal flag turned on. These history entries are in addition to those added by setting PICAHIST to Y. The application can control the creation of the real-time entries by setting the corresponding PIDT entry journal flags appropriately.

If PICAHIST is set to **Y** and the history data processing function is disabled, TSP BLGAPI05 or TSP BLGAPIPX sets a nonzero PICAREAS code. If the history data processing function is enabled, your application must be running under a privilege class that allows database administrator authority to set PICAHIST to **Y**. You can modify TSP BLGAPI05 or TSP BLGAPIPX to allow applications to set PICAHIST to **Y** to enable the history data processing function and to change the level of authority required to set PICAHIST to Y.

Date Considerations

When PICADFMT=0 (that is, X'00'), then any value specified in PICADSEP is ignored; that is, dates passed by your application to the API will be in the same format as they are in the SDDS portion of the database.

If you want to pass dates in a different format, specify that format in the PICADFMT and PICADSEP fields (PICADFMT is described on page 111 and PICADSEP is described on page 111). The API will convert the dates you pass into the default external date format specified in the session parameters before they are processed by Tivoli Information Management for z/OS. If you choose this option and your date format is longer than PIDTMAXL for a field, then set PIDTCURL to the length of your date. You will not receive a length error unless the date is longer than PIDTMAXL after it has been converted to thedefault external date format.

Field Specifications

You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T105.
PICARNID External record ID or root VSAM key of record to be updated.
PICAPIDT Pointer to the PIDT.

If you are using equal sign processing, you will need tospecify:

PICAEQRP Set to **Y**. If the response data (or visible phrase for direct-add items) contains an equal sign (=) then the data will be processed as equal data and processed according to the rules defined by the product.

When you are not using a dynamic PIDT, you must specify the following PIDT fields when processing responses:

PIDTDATP Pointer to data location in the response buffer (except for visible phrase and direct add items)

PIDTCURL Length of responses in the response buffer (except for visible phrase and direct add items)

PIDTCNFR Current number of responses for the field.

When you are using a dynamic PIDT, you can specify the following PIDT fields when processing responses:

PIDTDELO Set to **Y** to delete all other type (PIDTRDEF=O) entries from the record.

PIDTCHNG Set to **Y** to process this PIDT entry.

PIDTCURL Length of responses in the response buffer, if they have been changed (except for visible phrase and direct add items).

PIDTCNFR If the number of PIDT entries have changed, the current number of responses for the field.

PIDTMNCR If the number of PIDT entries have increased, the maximum number of responses for the field.

PIDTMAXL If the data in the PIDT have been increased, the maximum length of responses for the field.

PIDTDATP If PIDTMAXL or PIDTMNCR was increased and you moved the data to the free space area of the buffer.

You can specify how you want to process lists; that is, whether to update lists, append new list items to existing lists, or replace existing lists:

- PICALSTM = U (update) or A (append) or R (replace)

You must fill in certain fields when you specify text processing:

If you select buffer processing for text:

- PICATXTP = B
- PICATXAU = **Y** if text audit data is specified
- PICATXTR = **Y** if existing text is to be replaced
- PIDTDATP = pointer to text in the response buffer
- PIDTCURL = total text length
- PIDTCNFR = number of text units (lines) being processed

If you select data set processing for text:

- PICATXTP = D
- PICATXAU = **Y** if text audit data is specified
- PICATXTR = **Y** if existing text is to be replaced
- PIDTDATP = pointer to data set name in the response buffer
- PIDTCURL = length of data set name
- PIDTCNFR = 1

You must specify the following fields when processing history data:

- PICAHIST = **Y**
- PIDTPIHT = pointer to PIHT
- PIHTCURL > 0 for each data-entry to be added

If you choose equal sign processing:
- PICAEQRP = **Y**
- PIDTDATP = pointer to equal sign (=) in response buffer

You must specify the following field when processing a dynamic PIDT:
- PICADYNM=**Y** for dynamic PIDTs

You must specify the following field when using a root VSAM key in the PICARNID field:
- PICAVSAM=**Y**

You can specify values for these PICA fields if you want to change the name of the current application ID or the name of the current privilege class:

PICAUSRN The name by which Tivoli Information Management for z/OS recognizes your application.You can specify a value here to change the name of the current application ID. Tivoli Information Management for z/OS uses this name in place of a TSO user ID when performing record access privilege class processing. The value specified must be an eligible user of the current privilege class record.

PICACLSN A valid privilege class name. You can specify a value here to change the current privilege class record.

You can specify values for these PICA fields if you want to specify the format of dates your application will input to the LLAPI:

PICADFMT The index of the date format to use for exchanging datevalues between the API and your application.

PICADSEP The character slash (/)or hyphen (-) or period (.) used to separate month, day, and year portions of dates used in date formats which use a separator character.

Table 24 shows the update record transaction flow for a synchronous environment. It is assumed that update resources are obtained. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 24. LLAPI Transaction T105. Update Record (Synchronous)

Step	Program	Action
1	Application	■ Stores response data for each updated field in the response buffer. Each response can be no longer than the value in PIDTMAXL, and the number of responses (separated by the character in PIDTSEPC) cannot exceed the value in PIDTMNCR (except for list items). ■ When using a PIDT from the obtain record update resource transaction (T103): • Sets the buffer address in PIDTDATP. • Sets the length of the buffer in PIDTCURL. ■ When using a dynamic PIDT built by the retrieve record transaction (T100): • Sets PIDTDATP address if it is moved from the original location. • Sets PIDTCURL if the length increases or decreases from what it was initially. • Can alter PIDTMAXL and MIDTMNCR. ■ Sets PIDTCNFR to the current number of responses for a field.

Table 24. LLAPI Transaction T105 (continued). Update Record (Synchronous)

Step	Program	Action
		■ Sets PICA fields as follows: • PICATRAN=T105 (Update Record) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICARNID=ID or root VSAM key of record to update • PICAVSAM=Y if using a root VSAM key • PICADYNM=Y if using a dynamic PIDT • PICAPIDT=address of update PIDT or dynamic PIDT. • PICAUSRN=application ID if you want to change the name of the current application. • PICACLSN=privilege class if you want to change the privilege class. • PICALSTM= U to update list items or A to append new list items or R to replace list items ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Record Inquiry (T107)

This transaction searches the Tivoli Information Management for z/OS database by using entries in the PIDT and entries in the PIAT for search arguments. You must have a PIDT (either static or generated from a data view record) to start this transaction, but the type of search depends on whether you specify PIAT arguments. You can use a record type defining s-word to ensure that the resulting list is limited to a single record type, but you can omit the s-word for general inquiries.

Note: If you are using logical database partitioning (described in the*Tivoli Information Management for z/OS Program Administration Guide and Reference*), you should be aware that API applications cannot perform multipartition searches.

You can improve the performance of your application by managing how the LLAPI reads the database when performing an inquiry. If your application performs searches that result in very long search results lists, you can limit the length of the initial list and thus the amount of time your application requires to obtain the search results. Your application can then view other parts of the list as needed. You can save a search results list and retrieve records as you need them. See "Return of Selected Search Results" on page 87 for information on how to selectively return search results.

If you do not limit the number of matches returned, define your search arguments to limit the size of the returned PIRT to only what is necessary.

Note: Refer to the *Tivoli Information Management for z/OS User's Guide* for general information on searching the database.

Parenthetical searching

To increase your ability to eliminate unwanted records from the results of freeform searches, you can use parentheses within freeform search arguments to specify the order in which arguments should be evaluated. Arguments placed within parentheses will be evaluated first. The parentheses can adjoin the arguments or be separated by one or more spaces. The parentheses can be placed in the same PIAT row with the adjoining argument or can be in a separate PIAT row.

For example, the argument string

```
¬STAC/CLOSED (GROS/CEO | GROS/PAY) ¬(PRIO/03 | PRIO/04)
```

can be entered on separate PIAT rows like this:

```
¬STAC/CLOSED
(GROS/CEO
| GROS/PAY)
¬(PRIO/03
| PRIO/04)
```

or it can be entered on separate PIAT rows like this:

```
¬STAC/CLOSED
(
GROS/CEO
| GROS/PAY
)
¬(
PRIO/03
| PRIO/04
)
```

The argument can be entered in other ways as well, as long as the boolean operator (if one is present) appears first and no more than one argument is included in each PIAT row.

Date Considerations

When PICADFMT=0 (that is, X'00'), then any value specified in PICADSEP is ignored; that is, dates passed by your application to the API will be in the same format as they are in the SDDS portion of the database.

If you want to use a different date format, specify that format in thePICADFMT and PICADSEP fields (PICADFMT is described in 111 and PICADSEP is described in 111). The API will convert the dates you pass into the default external date format specified in the session parameters before they are processed by Tivoli Information Management for z/OS

and will convert dates in internal format in the database to your specified format before passing them to your application. If you choose this option and your date format is longer than PIDTMAXL for a field, PIDTCURL can exceed PIDTMAXL. You will not receive a length error unless the date is longer than PIDTMAXL after it has been converted to the default external format.

Return All Search Results

Follow these steps on the record inquiry process when you view all records in the search results list without saving the list:

1. Choose the static PIDT name or data view record ID that you want and put its name (static PIDT name or data view record ID) in PICATABN.
2. Specify, in PICAREQL, the size of the response buffer needed to contain structured arguments that simulate quick search field responses (must be greater than 0).
3. Determine whether freeform arguments are to be added to the structured arguments and specify the maximum number needed in PICAREQR.
4. Obtain inquiry resources (T106).
5. Store the structured arguments in the response buffer. See step 5 on page 64 for more information.
6. Store freeform arguments in the PIAT.

   Be sure to set the current number of arguments (PIATNARG) to the number used for this inquiry.
7. Optionally, specify an associated data field by putting the s-word index of the field in PICASRCH. This field must be defined in the PIDT you are using. An associated data field is a field extracted from each record and stored in the resultant match list. Your application must be prepared to process blanks in this field. Also be aware that the LLAPI cannot extract list item, phrase, and text data.
8. Perform the inquiry (T107). After the transaction finishes the search, check the value of PIRTCODE. If the value is not 00 in any PIRT row, then the LLAPI might have found record processing exceptions when trying to extract that record ID (RNID) or associated data from that record. The data in any PIRT row is unreliable if PIRTCODE is anything but 00 for that PIRT row.
9. Free the PIRT (T007) and PIDT (T006) if you are not going to make other inquiries.

Other Record Inquiry Considerations for All Search Results

1. If you are performing multiple searches on the same record type and you use different fields, reinitialize the used PIDT rows, or free record-inquiry resources (T006) and reallocate (T106) before storing data for the next search.
2. Consider using a session-parameters member that specifies a value for sort prefix SORTPFX-N1 that limits the number of results returned. Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for information on the BLGPARMS macro. When you specify SORTPFX-N1, the number of matches from the search is compared to the number in the SORTPFX-N1 field. PIRTSRRC is set to the number of matches from the search. The value in PIRTHITC is set according to the values in SORTPFX-N1 and PIRTSRRC:
   - If PIRTSRRC is less than the value in SORTPFX-N1, then PIRTHITC is set to the value in PIRTSRRC.

- If the value in PIRTSRRC is greater than or equal to the value in SORTPFX-N1, then PIRTHITC is set to the value in SORTPFX-N1.

At this point, the number of records read is based on the value in PIRTHITC. The number of records returned by the LLAPI is the smaller of the values in PIRTHITC and PICANUMH.

Return of Selected Search Results

Follow these steps on the record inquiry process to save results from a search:

1. Choose the static PIDT or data view record that you want and put its name (static PIDT name or data view record ID) in PICATABN.
2. Specify, in PICAREQL, the size of the response buffer needed to contain structured arguments that simulate quick search field responses (must be greater than 0).
3. Determine whether to add freeform arguments to the structured arguments and specify the maximum number needed in PICAREQR.
4. Obtain inquiry resources (T106).
5. Store the structured arguments in the response buffer. See step 5 on page 64 for more information.
6. Store freeform arguments in the PIAT.

   Be sure to set the current number of arguments (PIATNARG) to the number used for this inquiry.
7. Specify an associated data field by putting the s-word index of the field in PICASRCH. This field must be defined in the PIDT you are using. An associated data field is a field extracted from each record and stored in the resultant match list. Be prepared to accept blanks in this field. Also be aware that the API cannot extract list item, phrase, and text data.
8. To limit the size of the PIRT returned to your application, specify in PICANUMH the number of matches to return, and specify in PICABHIT the first match to return. These fields are optional.
9. Specify a 4-byte fixed search ID in PICASRID to save the search results. This field is required. If zeroes are specified, the search results are not saved.
10. Perform the inquiry (T107). When the transaction finishes, check the value of PIRTCODE. If the value is not 00 in any PIRT row, then the API might have found record processing exceptions when trying to extract that record ID (RNID) or associated data from that record. The data in any PIRT row is unreliable if PIRTCODE is anything but 00 for that PIRT row.
11. Free the PIRT (T007) and PIDT (T006) if you are not going to make other inquiries.

Follow these steps on the record inquiry process when you selectively view records from a previously saved search:

Note: You cannot specify new search criteria when retrieving records from a previously saved search.

1. To limit the size of the PIRT returned to your application, specify in PICANUMH the number of matches to return, and specify in PICABHIT the first match to return. These fields are optional.

2. To indicate the search from which you want to retrieve results, specify the 4-byte search ID in PICASRID. This field is required.
3. Specify **Y** in PICARHIT to return results from an existing search. This field is required.
4. Perform the inquiry (T107). When the transaction finishes, check the value of PIRTCODE. If the value is not 00 in any PIRT row, then the API might have found record processing exceptions when trying to extract that record ID (RNID) or associated data from that record. The data in any PIRT row is unreliable if PIRTCODE is anything but 00 for that PIRT row.
5. Free the PIRT (T007) if you are not going to make other inquiries.

Other Record Inquiry Considerations for Selected Search Results

See "Other Record Inquiry Considerations for All Search Results" on page 86 for information on considerations for selected search results.

Other Record Inquiry Considerations for All Searches

The following information applies when you either return all search results or save search results.

Argument Data Case Considerations

Free form arguments are used as entered and must be provided by the application in the proper case. Structured arguments are processed according to the setting of the **Cognize in mixed case?** option in the PIDT row or attribute record for the argument:

- If **Cognize in mixed case?** is **Y**
  - If validation is requested, the case of the argument (after any adjustments made of the validation module based on the setting of the **Collected data case** option) will be used for the search.
  - If validation is not requested, the case of the argument as passed by the application will be used for the search. No case transformation will be done.
- If **Cognize in mixed case?** is **N**
  - Upper case will be used for the search, regardless of the case passed by the application and regardless of any adjustments made of the validation routine.

Multiple or List Item Processing Considerations

When you want to search on multiple or list-item responses, the responses must be separated by the separator character specified in PIDT field PIDTSEPC. You cannot append a separator character to the last response of a field.

This is an example of a list item using a comma separator character:

```
moda,modb,modc
```

You can use data model records or static PIDTs to provide the view of the data for your application.If you use data model records, a PIDT is generated from the data view record and associated data attribute records. See "Field Validation Using the Field Validation Module BLGPPFVM" on page 279 for additional details on data model records.

The search arguments constructed are based on the argument data specified in the PIDT. You do not need a PIAT to start an inquiry, but a PIAT can be used to augment the search criteria. The LLAPI builds search arguments in the same order that the data occurs in the

tables. Arguments entered in the data response buffer with the PIDT entries cannot have any Boolean or range characters but can have an asterisk (*) or a period (.).

With the freeform argument search, you specify PIAT arguments in the sequence you want. PIATNARG specifies the number of arguments used for the search. Each row contains a specific argument. Your application can retrieve a prefix from the PIDT and append the argument data to it. If you are entering a range, put the first part of the range in one PIAT row and the second part of the range in the next PIAT row. The range character precedes the second part of the range. The LLAPI appends arguments entered in the PIAT to the arguments collected by the PIDT.

When the LLAPI returns the search results, the API builds a PIRT and returns it to the caller to indicate which record IDs contain instances of the search criteria. The caller can then read the records using the record retrieval transaction (T100).

The LLAPI stores associated data (limited to 45 characters) extracted from each record (if available) in each PIRT entry field (PIRTDATA). Your application stores the symbolic name of the field in PICASRCH before performing a search. This capability is similar to that provided on the interactive search results panel. An example of this process is to extract the description abstract information of a problem record by specifying symbolic index S0E0F. You cannot specify list entry, phrase, or text item data as an associated data field.

You must specify the following PICA fields to start this transaction:

PICATRAN A transaction code of T107.

PICAPIDT Pointer to inquiry PIDT. Your application must use an existing PIDT when requesting additional matches from an existing search. If your application specifies an existing search in PICASRID, this field is ignored.

If you are using equal sign processing, you will need tospecify:

PICAEQRP Set to **Y**. If the response data (or visible phrase for direct-add items) contains an equal sign (=) then the data will be processed as equal data and processed according to the rules defined by the product.

The following PICA fields are optional:

PICANUMH The maximum number of matches in the database returned from a search.

PICABHIT The beginning match number to return.

PICASRID The identifier of a search.

PICARHIT This field indicates to the API whether to return results from an existing search.

PICAUSRN The name by which Tivoli Information Management for z/OS recognizes your application.You can specify a value here to change the name of the current application ID. Tivoli Information Management for z/OS uses this name in place of a TSO user ID when performing record access privilege class processing. The value specified must be an eligible user of the current privilege class record.

PICACLSN A valid privilege class name.You can specify a value here to change the current privilege class record.

PICADFMT The index of the date format to use for exchanging datevalues between the API and your application.

PICADSEP The character slash (/)or hyphen (-) or period (.) used to separate month, day, and year portions of dates used in date formats which use a separator character.

You must set the following PIDT fields for structured search criteria:

PIDTDATP The address of responses in the response buffer
PIDTCURL The length of the responses
PIDTCNFR The current number of field responses

You must set the following PIAT fields for freeform search criteria:

PIATNARG The number of arguments to process
PIATDATL The length of the argument (for each argument)
PIATDATA The argument data item (for each argument)

The following PIRT field is optional:

PIRTBHIT The beginning match number to return.

Table 25 shows the record inquiry transaction flow for a synchronous environment assuming no freeform arguments. Search results are not saved. It is assumed that an inquiry resource is obtained. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 25. LLAPI Transaction T107. Record Inquiry without saving search results (Synchronous)

Step	Program	Action
1	Application	■ Stores response data for each inquiry field in the response buffer. ■ Sets the buffer address for each data field in PIDTDATP and its length in PIDTCURL. Each response can be no longer than the value in PIDTMAXL, and the number of responses, separated by the character in PIDTSEPC, cannot exceed the value in PIDTMNCR (except for list items). ■ Sets PIDTCNFR to the current number of responses for a field.

Table 25. LLAPI Transaction T107 (continued). Record Inquiry without saving search results (Synchronous)

Step	Program	Action
		■ Sets PICA and PIAT fields as follows: • PICATRAN=T107 (Record Inquiry) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies a received transaction's validity. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICAPIDT=address of inquiry PIDT • PICASRCH=associated data index (if applicable) • PICABHIT=specifies the first record to be returned from a search (optional) • PICANUMH=specifies the number of records returned from a search (optional) • PIATNARG=0 if PIAT obtained with resource (no freeform arguments in this example) • PICAUSRN=name of current application. • PICACLSN=name of current privilege class name. ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets following fields: • PICARETC • PICAREAS • PICAPIRT • PIRTHITC • PIRTBHIT • PIRTSRRC • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks following fields set by server: • PICARETC contains return code • PICAREAS contains reason code • PICAPIRT points to results table (PIRT) • PIRTHITC contains number of matches returned • PIRTSRRC contains number of matches found • PIRTBHIT contains the match index of the first match found. • PICAMSGC contains number of messages • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Table 26 on page 92 shows the record inquiry transaction flow to save and view initial search results for a synchronous environment assuming no freeform arguments. It is assumed that an inquiry resource is obtained. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 26. LLAPI Transaction T107. Record Inquiry to save and view initial search results (Synchronous)

Step	Program	Action
1	Application	■ Stores response data for each inquiry field in the response buffer. ■ Sets the buffer address for each data field in PIDTDATP and its length in PIDTCURL. Each response can be no longer than the value in PIDTMAXL, and the number of responses, separated by the character in PIDTSEPC, cannot exceed the value in PIDTMNCR (except for list items). ■ Sets PIDTCNFR to the current number of responses for a field. ■ Sets PICA and PIAT fields as follows: • PICATRAN=T107 (Record Inquiry) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies a received transaction's validity. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICAPIDT=address of inquiry PIDT • PICASRCH=associated data index (if applicable) • PICASRID=identifier for this search • PICARHIT=a value other than **Y** indicating to not return exiting matches • PICABHIT=specifies the first record to be returned from a search (optional) • PICANUMH=specifies the number of records returned from a search (optional) • PIATNARG=0 if PIAT obtained with resource (no freeform arguments in this example) • PICAUSRN=name of current application. • PICACLSN=name of current privilege class name. ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets following fields: • PICARETC • PICAREAS • PICAPIRT • PIRTHITC • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks following fields set by server: • PICARETC contains return code • PICAREAS contains reason code • PICAPIRT points to results table (PIRT) • PIRTHITC contains number of matches returned • PIRTSRRC contains number of matches found • PIRTBHIT contains the match index of the first match found. • PICAMSGC contains number of messages • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Table 27 shows the record inquiry transaction flow to view existing search results for a synchronous environment. It is assumed that the search was previously performed and the results saved. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures" on page 100.

Table 27. LLAPI Transaction T107. Record Inquiry to view existing search results (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA and PIAT fields as follows: • PICATRAN=T107 (Record Inquiry) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies a received transaction's validity. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICASRID=identifier of search previously saved • PICABHIT=specifies the first record to be returned from a search (optional) • PICANUMH=specifies the number of records returned from a search (optional) • PICARHIT=R to return existing matches • PICAUSRN=name of current application. • PICACLSN=name of current privilege class name. ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets following fields: • PICARETC • PICAREAS • PICAPIRT • PIRTHITC • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks following fields set by server: • PICARETC contains return code. • PICAREAS contains reason code. • PICAPIRT points to results table (PIRT). • PIRTHITC contains number of matches returned • PIRTSRRC contains number of matches found • PIRTBHIT contains the match index of the first match found. • PIRTHITC contains number of matches found. • PICAMSGC contains number of messages. • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Add Record Relation (T109)

This transaction adds record relations to Tivoli Information Management for z/OS records.

You use this transaction to create a relationship between a parentrecord and child records. For example, you can link a change record to change activity records. The transaction updates the parent record and adds nonreplaceable data items to the record. Specifically, you add child relations to the parent records identified by index number in the following list:

S0B06 Add activity record names to a change record.

S0B0F
Add feature record names to a configuration hardware component record.

S0B13 Add feature record names to a configuration software component record.

S0B0F
Add connected-to record identifiers to a hardware component record.

S0B13 Add connected-to record identifiers to a software component record.

Note: This is the only LLAPI transaction that adds nonreplaceable data to the database.

You can use data model records or static PIDTs to provide the view of the data for your application.If you use data model records, a PIDT is generated fromthe data view record and associated data attribute records. See “Field Validation Using the Field Validation Module BLGPPFVM” on page 279 for additional details on data model records.

Note: If you are using logical database partitioning, you can perform an add record relation to a record only if the Owning Partition of that record matches the Primary Partition of your privilege class.

Special static PIDTs are provided with the LLAPI. These tables specify the related record prefix data used to store related record names. See “Record Type and Function PIDT Tables” on page 299 for more information on these PIDTs.

You can use this transaction to store record relations in complex panel set data models. Refer to the *Tivoli Information Management for z/OS Panel Modification Facility Guide* for information about complex panel sets.

Checking out the record before the update ensures that no other users can update the record prior to your update. Your administrator can define a time limit for checked out records (in the BLX-SP parameter APICHKOUTLIM, described in the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*) so that records will not inadvertently remain indefinitely checked out if your application does not check in the record.

You use the response data buffer (used with a particular related record PIDT pointed to by PICA field PICAPIDT) to specify the related record names or identifiers prefixed and collected in the parent record specified in PICA field PICARNID. If you specify more than one name or identifier, they must be separated by the separator character set in PIDT field PIDTSEPC.

To add relations to a parent record, use the following transactions in the order given.

T101 Obtain the PIDT resource for child record creation.

T104 Check out the parent record to which relations are to be added.

T102 Create the child record or records.

T006 Free the create child PIDT.

T108 Obtain the add record relation resource for adding child names to the parent record.

T109 Add the record relations to the parent record.

T008 Check in the parent record.

T006 Free the add record relation PIDT.

Example

Here is the exact same list of transactions, with the exception that this list uses the example of adding activities named ACT1 and ACT2 to a change record, CHG1.

T101 Obtain the PIDT resource for activity record creation (BLGYACC).
T104 Check out the change record CHG1 to which activities are to be added.
T102 Create the activity records called ACT1 and ACT2.
T006 Free the create activity PIDT (BLGYACC).
T108 Obtain the add record relation resource for adding activity names to the change record (BLGYCHA).
T109 Add the record relations to the change record, CHG1. If another application or user is attempting to update the record, the record might be unavailable. You can direct the LLAPI to either retry this transaction from 1 to 255 times before returning control to your application or wait until the record is available. See page 18 for more information.
T008 Check in the change record, CHG1. If another application or user is attempting to update the record, the record might be unavailable. You can direct the LLAPI to either retry this transaction from 1 to 255 times before returning control to your application or wait until the record is available. See page 18 for more information.
T006 Free the add record relation PIDT (BLGYCHA).

You must specify the following PICA fields to start this transaction:
PICATRAN A transaction code of T109
PICARNID External record ID of parent record
PICAPIDT Address of add relation PIDT

If you are using equal sign processing, you will need tospecify:

PICAEQRP Set to **Y**. If the response data (or visible phrase for direct-add items) contains an equal sign (=) then the data will be processed as equal data and processed according to the rules defined by the product.

You can specify values for these PICA fields if you want to change the name of the current application ID or the name of the current privilege class:

PICAUSRN The name by which Tivoli Information Management for z/OS recognizes your application.You can specify a value here to change the name of the current application ID. Tivoli Information Management for z/OS uses this name in place of a TSO user ID when performing record access privilege class processing. The value specified must be an eligible user of the current privilege class record.

PICACLSN A valid privilege class name.You can specify a value here to change the current privilege class record.

Table 28 on page 96 shows the add record relation transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see “LLAPI Structures” on page 100.

Table 28. LLAPI Transaction T109. Add Record Relation (Synchronous)

Step	Program	Action
1	Application	■ Stores response data for each updated field in the response buffer. Sets the buffer address in PIDTDATP and its length in PIDTCURL. Sets PIDTCNFR to the current number of responses for a field. ■ Sets PICA fields as follows: • PICATRAN=T109 (Add Record Relation) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICARNID=parent record ID • PICAPIDT=address of add relation PIDT • PICAUSRN=application ID if you want to change the name of the current application. • PICACLSN=privilege class if you want to change the privilege class. ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code • PICAREAS contains reason code • PICAMSGC contains number of messages • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Delete Record (T110)

This transaction deletes a record. Your application specifies the external record ID or the root VSAM key of the record to be deleted in PICA field PICARNID. You can delete records of all types.

You must specify the following PICA fields to start this transaction:

PICATRAN Transaction code of T110.

PICARNID External record ID or root VSAM key of record to delete. You must also specify the following field when using a root VSAM key in PICARNID:
■ PICAVSAM = **Y**

You can specify values for these PICA fields if you want to change the name of the current application ID or the name of the current privilege class:

PICAUSRN The name by which Tivoli Information Management for z/OS recognizes your application.You can specify a value here to change the name of the current application ID. Tivoli Information Management for z/OS uses this name in place of a TSO user ID when performing record access privilege class processing. The value specified must be an eligible user of the current privilege class record.

PICACLSN A valid privilege class name.You can specify a value here to change the current privilege class record.

Root VSAM Key Considerations

When PICAVSAM is set to Y, the LLAPI attempts to process the record using the record ID. TSP BLGAPI10 calls user exit BLGYAPBU to retrieve the record ID. If the record ID can be determined, the record is deleted with the delete or purge command. If the record ID cannot be determined because the record ID cannot be read (the record is damaged), the root VSAM key is used to delete the record only if this function has been enabled. If the function has not been enabled or if the record ID cannot be determined for a reason other than a damaged record (such as a duplicate record ID error), TSP BLGAPI10 sets a nonzero PICAREAS code and does not attempt to delete the record. If the function has been enabled, your application must be running under a privilege class that allows database administrator authority to allow deletion using the VSAM root key. TSP BLGAPI10 can be modified to allow applications to set PICAVSAM to **Y** and delete VSAM records with any level of authority you want or for any error other than a damaged record error.

If the record is damaged, the normal delete processing is unable to complete. You must run the SDIDS build utility, BLGUT1, to correct the SDIDS. Until the utility is run, the record shows as “deleted” if it appears on a search results list.

Note: If you are using logical database partitioning, you can delete a record only if the Owning Partition of that record matches the Primary Partition of your privilege class.

Table 29 on page 98 shows the delete record transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see “LLAPI Structures” on page 100.

Table 29. LLAPI Transaction T110. Delete Record (Synchronous)

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T110 (Delete Record) **Note:** You can use this transaction asynchronously if you initialized Tivoli Information Management for z/OS (T001) in asynchronous mode. Asynchronous mode operation returns control from the server to the application as soon as the server verifies the validity of a received transaction. The application can then check the return code and, if no error is detected, perform other processes. The application can periodically check the status of the transaction being processed by the server, or it can start the sync and wait on completion transaction (T009) and wait for the transaction being processed by the server to complete. See explanations of T009 and T010 check and sync transactions on page 39. • PICARNID=record ID or root VSAM key of record to be deleted • PICAVSAM=Y if using a root VSAM key. • PICAUSRN=application ID if you want to change the name of the current application. • PICACLSN=privilege class if you want to change the privilege class. ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICARNID • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code • PICAREAS contains reason code • PICARNID contains 0 • PICAMSGC contains number of messages • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

Change Record Approval (T112)

This transaction provides a means to approve or reject a change record. By using this transaction, you can pass approvals from another change management product or application, or from a Web application into Tivoli Information Management for z/OS. This is similar to the process to approve or reject changes that you can do interactively; additional information on the interactive process to perform this function can be found in the *Tivoli Information Management for z/OS Problem, Change, and Configuration Management* document. Your application must specify the name of a privilege class approving or rejecting the change and specify whether to approve or reject the change.

A specified change record is updated as follows:

- If approval status is specified as “accepted”, the current privilege class in the list of approvers within the change record is marked as “approval accepted”.

- If the status is specified as “rejected” or not “accepted”, the current privilege class in the approver list is marked as “approval rejected”.
- When one approver rejects the change, the change record is marked as “rejected”.
- When all of the approvers on the list have accepted the change, the change record is marked as “accepted”.
- Before the change record is marked “accepted” or “rejected”, it is in the “approval pending” status.

Note: If data attribute records are used as direct add fields, then normal file processing is not performed for change records when change approval processing is being performed. That is, if ALL of these five direct adds—DATE/, TIME/, CLAE/, DATM/, and TIMM/—are changed to data attribute records, then data modified, time modified, and user ID are not saved in the record

To use the change record approval transaction, perform the following actions:

PICATRAN Set this to a transaction code of T112.

PICARNID Specify the external record ID of the change record.

Provide your authorization to perform the approval transaction:

PICATABN Set this to the alias or member name of a static PIDT used to retrieve change records. A data view record can be used in place of a static PIDT. To do so, set PICATABN to the name of the data view record and set PICADMRC=Y. Ensure that the PIDT or data view record specifies the authorization code for displaying change records.

Specify the desired approval status for the change record:

PICACHAP If you want to specify an approval status of “accepted”, set PICACHAP=A; if no approval status is specified or if the status is not “accepted” (PICACHAP≠A), then the default is to reject the approval of the change record.

Specify the privilege class of the approver:

PICACLSN If no privilege class is set with this transaction, then the default is to use the privilege class that is currently in effect. If you want to change to a different privilege class, provide a value for PICACLSN. Ensure that the privilege class has authority to display change records.

Specify the application ID of the approver:

PICAUSRN If no application ID is set with this transaction, then the default is to use the application ID that is currently in effect. If you want to change to a different application ID, provide a value for PICAUSRN. The user must be an eligible user of the privilege class specified in the PICACLSN.

PICAAPVR Set this value to the approver. This can be a different value than PICACLSN or the current profile class. If no approver is set with this transaction, then the default is to use the value specified by PICACLSN. If PICACLSN is not set, the privilege class that is currently in effect is used as the default.

Table 30 shows the change record approval transaction flow for a synchronous environment. For more detailed information on the LLAPI structures and their fields, see "LLAPI Structures".

Table 30. LLAPI Transaction T112. Change Record Approval

Step	Program	Action
1	Application	■ Sets PICA fields as follows: • PICATRAN=T112 (Change Record Approval) • PICARNID=record ID of the change record to be approved or rejected • PICATABN=name of a static PIDT or data view record with the appropriate authority • PICADMRC=Y if PICATABN is the name of a data view record • PICACHAP=A for approve; any other value results in reject • PICAUSRN=application ID (if you want to change the name of the current application) • PICACLSN=privilege class (if you want to change the privilege class; if you change the privilege class, this has the effect of rejecting the change.) ■ Calls BLGYSRVR(*PICA*).
2	Server	■ Validates PICA fields ■ Notifies API subtask ■ Waits for completion ■ Sets the following PICA fields: • PICARETC • PICAREAS • PICAMSGC • PICAMSGP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • PICARETC contains return code • PICAREAS contains reason code • PICAMSGC contains number of messages • PICAMSGP points to message chain if PICAMSGC > 0. ■ Continues processing.

LLAPI Structures

The LLAPI uses several program structures to support the transactions your application uses to access the Tivoli Information Management for z/OS database. These structures are:

- Low-Level program interface communications area (PICA)
- Program interface alias table (PALT)
- Program interface data table (PIDT)
- Program interface history table (PIHT)
- Program interface pattern table (PIPT)
- Program interface argument table (PIAT)
- Program interface results table (PIRT)
- Program interface message block (PIMB)

Figure 5 on page 101 shows the relationships between the PICA control block structure and the other structures the LLAPI uses.

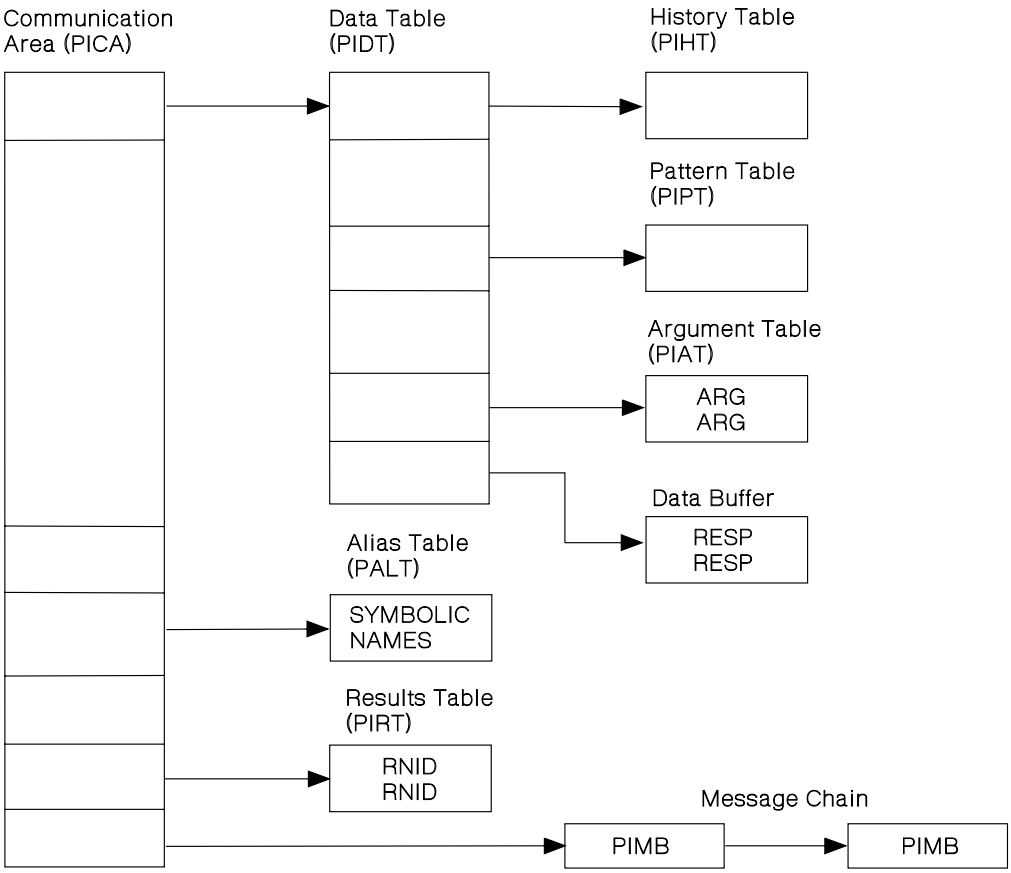

Figure 5. Relationships Between the PICA and the Interface Tables

Low-Level Program Interface Communications Area (PICA)

Your application allocates the PICA. The PICA is used to communicate between the rest of the interface (including the API subtask) and your application. The PICA also serves as an anchor to all other LLAPI structures. You can find a sample PICA in the macro library of Tivoli Information Management for z/OS (SBLMMACS). Look for BLGUPICA.

Table 31 shows the structure of the PICA and the page number where the table fields are explained.

Note: As shown in the table, some fields are set by the interface. Your application should *not* attempt to set these fields. If it does, results are unpredictable.

Table 31. LLAPI Communications Area (PICA)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
PICAACRO	0(0)	4	The acronym PICA (character)	Application	104
PICALENG	4(4)	4	Length of this structure (fixed)	Application	104
PICAENVP	8(8)	4	Transaction environment anchor (pointer)	Interface	104
PICASESS	12(C)	8	Session-parameters member name (character), minimum of 7 characters	Application	104
PICAUSRN	20(14)	8	Application ID (character)	Application	104

Table 31. LLAPI Communications Area (PICA) (continued)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
PICATRAN	28(1C)	4	Transaction code (character)	Application	104
PICACLSN	32(20)	8	Privilege class name (character)	Application	104
PICACLSC	40(28)	4	Privilege class record count (fixed)	Application	105
PICARETC	44(2C)	4	Transaction return code (fixed)	Interface	105
PICAREAS	48(30)	4	Transaction reason code (fixed)	Interface	105
PICAMSGC	52(34)	4	Message block count (fixed)	Interface	105
PICAMSGP	56(38)	4	Address of message chain (pointer)	Interface	105
PICARNID	60(3C)	8	Tivoli Information Management for z/OS record ID or root VSAM key (character)	Either	105
PICAPIDT	68(44)	4	Address of PIDT (pointer)	Either	105
PICAPIRT	72(48)	4	Address of PIRT (pointer)	Either	106
PICAREQR	76(4C)	4	Requested size of PIAT in rows (fixed)	Application	106
PICAREQL	80(50)	4	Length of requested response buffer (fixed)	Application	106
PICATINT	84(54)	4	Transaction time interval (fixed)	Application	106
PICASPLI	88(58)	4	Spool time interval (fixed)	Application	106
PICATABN	92(5C)	8	Static PIDT name (character - last position blank; dynamic PIDT, * appended) or record ID of a data view record	Either	106
PICADBID	100(64)	1	Database ID (character)	Application	106
PICASTXT	101(65)	1	Suppress text indicator (character Y)	Application	106
PICAASYN	102(66)	1	Asynchronous environment indicator (character Y)	Application	107
PICASRCH	103(67)	5	Inquiry associated data index (character)	Application	107
PICARTIV	108(6C)	4	Residual time interval (unsigned fixed)	Interface	107
PICAMSGD	112(70)	1	Message destination indicator (character P, C, B)	Application	107
PICAVSAM	113(71)	1	VSAM sequence number indicator (character Y)	Application	107
PICAHIST	114(72)	1	Process history indicator (character Y)	Application	108
PICATXTR	115(73)	1	Replace or delete text on update indicator (character Y)	Application	108
PICAPARM	116(74)	4	User TSP parameter (pointer)	Application	108
PICATBLN	120(78)	8	Alias table name (character)	Application	108
PICATBLP	128(80)	4	Address of alias table (pointer)	Either	108
PICATXTU	132(84)	4	Maximum number of text units to retrieve (fixed). Default = 60.	Application	108
PICATXTW	136(88)	4	Maximum text unit width to retrieve (fixed). Default = 60.	Application	108
PICATXTP	140(8C)	1	Type of text processing to perform (character). B = buffer, D = data set. Default = D.	Application	108

Table 31. LLAPI Communications Area (PICA) (continued)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
PICATXTA	141(8D)	1	Area of text units (character). T = top block, B = bottom block. Default = B.	Application	108
PICATXAU	142(8E)	1	Audit data specified indicator (character Y)	Application	109
PICADYNM	143(8F)	1	Dynamic PIDT request on retrieve indicator (character Y)	Application	109
PICASTPA	144(90)	4	Address of subtask TCB address (pointer)	Interface	109
PICAESPC	148(94)	4	Extra PIDT entry space	Application	109
PICASRID	152(98)	4	Search ID for saved searches (fixed)	Application	109
PICANUMH	156(9C)	4	Number of hits to return for a search (fixed)	Application	109
PICABHIT	160(A0)	4	Index of first search match to return (fixed)	Application	109
PICARHIT	164(A4)	1	Return existing search hits (character Y)	Application	109
PICAHMEM	165(A5)	1	Use memory above 16MB indicator (character Y)	Application	110
PICAEQRP	166(A6)	1	Equal processing indicator	Application	110
PICADRIF	167(A7)	1	Bypass panel processing indicator	Application	110
PICADMRC	168(A8)	1	Data model record indicator	Application	110
PICADFMT	169(A9)	1	Date format indicator	Application	111
PICADSEP	170(AA)	1	Date separator character indicator	Application	111
PICACHAP	171(AB)	1	Change record approval status indicator (A=Accept; any other character indicates Reject)	Application	111
PICARSV2	172(AC)	8	Reserved. Must be initialized to binary zeros.	Application	111
PICAUTSP	180(B4)	8	Name of TSP or TSX to invoke on T111 (character)	Application	111
PICAPARL	188(BC)	2	If =0, indicates that the value contained in PICAPARM (X'74') is the address of a user buffer; if greater than 0, indicates that the value contained in PICAPARM is the address of a string in which case the value specified in PICAPARL is the length of the string being passed (fixed)	Application	112
PICALSTM	190(BE)	1	Update list processing mode	Application	112
PICASAUD	191(BF)	1	Suppress text audit data indicator (character Y)	Application	112
PICARSV3	192(C0)	12	Reserved. Must be initialized to binary zeros.	Application	112
PICATZON	204(CC)	8	TIMEZONE value	Application	112
PICAAPVR	212 (D4)	8	Approver	Application	112
PICATBFL	220 (DC)	4	Length of text argument buffer	Application	112
PICATBUF	224 (E0)	4	Address of text argument buffer	Application	112
PICARSV4	228 (E4)	28	Reserved. Must be initialized to binary zeros.	Application	112

The following list describes the purpose of each field of the PICA.

PICAACRO
A 4-character field containing the character string PICA to identify this as a communication area. After allocating storage for this structure, your application sets this field to the string PICA. The API checks for this character string at this location when each transaction begins.

PICALENG
A 4-byte fixed field containing the length of the PICA structure. The value in this field represents the total size of this structure including the PICAACRO field. Your application sets this field and the API validates it.

PICAENVP
A 4-byte pointer field containing the address of the LLAPI environment area.

Note: Initialize this pointer field to zero when it is passed to the LLAPI for the first time during an initialize Tivoli Information Management for z/OS (T001) transaction.

Your application must maintain the address stored in this pointer until the terminate Tivoli Information Management for z/OS (T002) transaction is complete.

PICASESS
An 8-character field containing a 7- or 8-character session-parameters member name used by the initialization routines when your application uses a T001 transaction to initialize Tivoli Information Management for z/OS. You can use the session-parameters member to specify unique data tables (specified in BLGFMT) and panel data sets for your application's use. This field is processed only during API initialization. Your application sets this field.

PICAUSRN
A 1- to 8-character name that identifies your application to Tivoli Information Management for z/OS. The name specified in this field is used in place of a TSO user ID when performing privilege class processing. The name you specify here must be an eligible user ID in the privilege class specified in the PICACLSN. If you are using APISECURITY=ON keyword in the BLX-SP startup parameters member, you must ensure that the MVS user ID(s) running this application are allowed to use this application ID. See "API Security" on page 287 for additional information. Your application sets this field.

PICATRAN
A 4-character transaction code that specifies a transaction service provided by the API. Your application sets this field.

PICACLSN
A 1- to 8-character privilege class name used when executing a transaction. The class name specified here must contain the authority needed to perform the requested Tivoli Information Management for z/OS record processing function. For example, if you are doing a problem display, the current privilege class must be one that permits problem display. Your application sets this field before it starts the initialize Tivoli Information Management for z/OS transaction (T001) and can set this field for any other transaction before processing that transaction. Every time you start a

transaction, Tivoli Information Management for z/OS compares the value in this field to the current privilege class. If the values are different, Tivoli Information Management for z/OS starts the class listed in this field. Once you specify a privilege class name with this field, it remains in effect until your application changes it. To change the privilege class for a future transaction, you must reset this field.

This field can contain mixed data. If it does, the API validates the field to make sure it contains valid mixed data.

PICACLSC
A 4-byte fixed field containing the maximum number of privilege class records that can be held in storage in the current session. When the LLAPI reaches the class count limit and a new class record is required, the least recently used class record is removed to make room for the new class record. By specifying a count, you reduce the number of record I/Os needed when your application requests multiple class records during transaction processing. This field is processed only during an initialize Tivoli Information Management for z/OS transaction (T001). Your application sets this field.

PICARETC
A 4-byte fixed return code field. The API sets this field. See “Return and Reason Codes” on page 301 for a list of return codes.

PICAREAS
A 4-byte fixed reason code field. The API sets this field. See “Return and Reason Codes” on page 301 for a list of reason codes.

PICAMSGC
A 4-byte fixed field that contains the count of messages in the message chain. The API sets this field.

PICAMSGP
A 4-byte pointer to a chain of messages associated with a transaction. This field contains zeros when the PICA field PICAMSGC is also zero. The API sets this field and maintains storage for this chain. Your application must not alter this pointer field.

PICARNID
An 8-character external record ID or root VSAM key used to identify which Tivoli Information Management for z/OS record the API is processing. Your application or the API sets this field, depending on the transaction requested.

When your application sets this field, the API will validate the field to ensure it contains valid mixed data.

When using a root VSAM key, enter it as it is listed on panel BLG1TVID. Refer to the *Tivoli Information Management for z/OS Diagnosis Guide* for details on locating the root VSAM key. For example, if the root VSAM key is X'0000001F', in a C language program you would specify PICARNID='0000001F'.

PICAPIDT
A 4-byte pointer to a PIDT used with a transaction. This field is required for all record access and search operations. Your application or the API sets this field, depending on the transaction requested.

PICAPIRT

A 4-byte pointer to a PIRT provided by the API to use with the record inquiry transaction (T107). Your application or the API sets this field.

PICAREQR

A 4-byte fixed field containing the requested size of the PIAT in rows. Tivoli Information Management for z/OS processes this field only for an Obtain Inquiry Resources transaction (T106). The API stores the argument table storage address in the PIDT header at field PIDTPIAT. When this field is zero, no PIAT is allocated with the PIDT. Your application sets this field.

PICAREQL

A 4-byte fixed field containing the size of the response buffer required when requesting create (T101), update (T103), inquiry (T106), or add record relation (T108) resources. The API stores the address and length of the response buffer in PIDT fields PIDTBUFP and PIDTBUFL. Your application sets this field. For dynamic PIDTs, your application can set this field on the retrieve (T100) transaction when requesting a dynamic PIDT (PICADYNM=Y). This amount is added as free space to the end of the response buffer built for the dynamic record retrieve. If your application sets this field on the retrieve transaction and PICADYNM≠Y, this field is ignored.

PICATINT

A 4-byte fixed field containing the number of seconds in which transactions must complete before your application is notified for further action (for asynchronous processing) or before the transaction is terminated (for synchronous processing). If you specify a value less than 45 seconds, the value is set to 45 seconds by default. If you specify a value of 0 or omit this field, the value is set to 300 seconds (five minutes) by default. See the sync transaction (T009), page 39 for more information on the use of this time interval. Your application sets this field only for an initialize Tivoli Information Management for z/OS transaction (T001).

PICASPLI

A 4-byte fixed field containing the time interval (in minutes) between instances where the activity log is spooled and reallocated if messages are being printed. The field is referenced only during an initialize Tivoli Information Management for z/OS transaction (T001). Your application sets this field.

PICATABN

An 8-character field containing a PIDT name. This field is made up of the 7-character PIDT name right-padded with a blank. Your application sets this field. For a dynamic PIDT, the LLAPI appends an asterisk (*) to the name. If you are using data model records (**PICADMRC=Y**), then the value specified for PICATABN is the data view record. A static PIDT name can be from 1 to 7 characters in length. A data view record ID can be from 1 to 8 characters in length. The API or your application sets this field.

PICADBID

A 1-character database ID identifying the database of the record to process. For Tivoli Information Management for z/OS records, the database ID value can be 4, 5, 7, 8 or 9. To specify the Tivoli Information Management for z/OS database, use a value of 5. Your application sets this field.

PICASTXT

A 1-character field that indicates to the API whether or not text data is to be

retrieved. Transactions that suppress text data use fewer resources and operate faster than those that do not. This field should be set to **Y** if text suppression is requested. Any other value indicates that text is to be returned. Your application sets this field.

PICAASYN

A 1-character field indicating whether the API should run in asynchronous mode or not. Asynchronous run mode causes the API to return to the application immediately after receiving a transaction request. A synchronize transaction (T009) or check transaction (T010) must be paired with every transaction attempted by the application when running in this mode. This field is processed only by an initialize transaction (T001). The field should be set to **Y** if asynchronous mode is desired. If this field has any other character value, it is ignored, and synchronous mode is used. Your application sets this field.

PICASRCH

A 5-character index value (the value in PIDTSYMB) of the data item to be retrieved (along with the record ID) and stored in the PIRT field PIRTDATA when your application starts an inquiry transaction (T107). This field is optional. Your application sets or clears this field.

Note: The API does not return list item, phrase, or text data in this field.

PICARTIV

A 4-byte unsigned binary field containing the time remaining in the interval specified by PICATINT in units of 26.04166 microseconds. If a transaction timeout does not occur, this value reflects the time that remains for the last time interval. You can use this field to determine transaction processing time. The calculation to determine length of time in minutes is:

```
PICATINT - ((((PICARTIV value) * 26.04166) / 1,000,000) / 60) = minutes
```

This field is valid only when transactions have completed. The API sets this field.

PICAMSGD

A 1-character field indicating destination of messages produced by the API subtask. The character options and their meanings are:

P Return output messages to the data set identified by the APIPRINT DD statement for the job step in which the API application is running. If this is a batch job, the JCL for the step where your application runs must have a DD statement similar to this:

```
//APIPRINT DD DSN=userid.API.OUTPUT,DISP=OLD
```

If your application is being run interactively, the file APIPRINT must be allocated prior to invoking the application.

C Return output messages on the message chain.

B Perform the functions of both P and C.

This field is processed only by an initialize transaction (T001). Any characters other than P or B are treated as C. Your application sets this field.

PICAVSAM

A 1-character field indicating whether the PICARNID field contains a root VSAM key or a record ID to identify the Tivoli Information Management for z/OS record

the API is processing. A **Y** indicates that the root VSAM key is used. Any other value indicates that the record ID is used. Your application sets this field.

PICAHIST
A 1-character field indicating whether history entries are to be processed by the retrieve, create, and update transactions. A value of **Y** indicates that history entries are to be processed. Any other value indicates they are not to be processed. Your application sets this field.

PICATXTR
A 1-character field indicating whether to replace or delete existing text on the update transaction. A value of **Y** indicates to replace or delete existing text. Any other value indicates not to replace or delete existing text. When the value is **Y**, a single separator character as data for a text row indicates to delete existing text of that type. When the value is **Y**, any text data other than a single separator character replaces existing text of that type in the record. Your application sets this field.

PICAPARM
A 4-byte pointer field that specifies the address of a parameter when your application starts a user Terminal Simulator Panel (TSP) with a T111 transaction. See "Start User TSP or TSX (T111)" on page 52 for more information about invoking a user TSP and the use of this parameter value. Your application sets this field.

PICATBLN
An 8-character field containing the name of the alias table (PALT). The name is from 1 to 8 characters long, right-padded with blanks. Your application sets this field.

PICATBLP
A 4-byte pointer to an alias table (PALT) used with transactions T011 and T012. When using T011 to get an alias table, the API sets this field with the address of the obtained table. When using T012 to free the alias table resource, your application sets the field with the address of the table resource to be freed.

PICATXTU
A 4-byte fixed field containing the maximum number of text lines to retrieve for each text item in a record when performing buffer processing. If no value is specified, a default value of 60 is used. Your application sets this field.

PICATXTW
A 4-byte fixed field containing the maximum width of a text line to be retrieved when performing buffer processing. Your application can specify a value from 1 to 132. If it does not specify a value, the API assigns a default value of 60 to this field.

PICATXTP
A 1-byte character field containing the type of text processing you want performed. A value of D in this field specifies that the LLAPI stores retrieved text in a data set. A value of B in this field specifies that the LLAPI stores retrieved text in the response buffer. If a blank or other character is specified, then a default value of D is used. Your application sets this field.

PICATXTA
A 1-byte character field indicating whether the LLAPI processes the top or bottom area of text lines when the number of lines available exceeds the value in PICATXTU. A value of T in this field specifies that the LLAPI processes the top

area of lines. A value of B in this field specifies that the LLAPI processes the bottom area of lines. If you specify a blank or another character, then the LLAPI uses a default value of B. Your application sets this field.

PICATXAU
A 1-character field indicating whether audit (or control) data is specified with input text on the create and update transactions. A value of **Y** indicates that each line of incoming text contains audit data (in the same form that is returned using the retrieve transaction). Any other value indicates that incoming text does not contain audit data. Your application sets this field.

PICADYNM
A 1-character field indicating whether a dynamic PIDT is to be generated on the retrieve record transaction (T100) or used on a create record (T102) or update record (T105) transaction. A value of **Y** indicates that a dynamic PIDT is to be processed. Any other value in this field indicates that a dynamic PIDT is not to be processed. Your application sets this field.

PICASTPA
A 4-byte pointer to the address storing the address of the subtask TCB. The field pointed to contains either the address of the subtask TCB (if subtask is active) or zero (if subtask is inactive). The API sets this field.

Note: If your application uses an ESTAE exit, you might need to detach the Tivoli Information Management for z/OS subtask. Before issuing a DETACH, your application should first check for a subtask TCB address.

PICAESPC
A 4-byte fixed field containing an amount of extra storage to be added to each entry buffer of a dynamic PIDT beyond what is needed to hold the data for the entry. This field is valid for dynamic PIDTs only. Your application sets this field.

PICASRID
A 4-byte fixed field containing the identifier of a search. It is assigned to either a new search results list or an existing search results list. If the value of this field is zero, the search results are not saved.

PICANUMH
A 4-byte fixed field containing the maximum number of matches in the database returned from a search:

- If this field is blank, either the value in SORTPFX-N1 from the session-parameters member or the actual number of hits is used, whichever is smaller.
- If this field is larger than the value in SORTPFX-N1, the value in SORTPFX-N1 is used.
- If this field is larger than the actual number of matches from the search, the actual number of matches is used.

PICABHIT
A 4-byte fixed field containing the beginning match number to return. If your application specifies zero, the API uses a value of one.

PICARHIT
A 1-character field indicating how the API treats this search. If this field is set to Y,

it indicates to the API to return results from an existing search. If this field is not set to Y, the API treats this search as a new search.

PICAHMEM

A 1-character field that indicates to the API whether or not control blocks may be returned to the application program in memory obtained above the 16MB address range. A value of **Y** indicates that memory above the 16MB address range may be used. Any other value indicates that the memory must be below the 16MB address range. Your application sets this field.

PICAEQRP

Equal processing indicator. This is set by your application. A **Y** indicates that an equal sign in the first character of the response data (or visible phrase for direct-add items) should be processed as equal data and processed according to the rules defined by the product. Any other value indicates that the equal sign should be treated as data; equal processing is not performed.

- For PIDT entry type R (response data), if the user passes an equal sign and PICAEQRP is not set to **Y**, then processing remains unchanged. The equal sign is used as data. If PICAEQRP is set to **Y**, and no equal pattern exists for that field, then the equal sign is used as data.
- If the PIDT entry type is D (direct add data), and the data contains an equal sign but PICAEQRP is not set to **Y**, then the equal sign data is used as data and entered into the record.

PICADRIF

Bypass panel processing indicator. This is set by your application. A **Y** indicates that no panels other than those used by the delete transaction should be used in record processing. Any other value indicates panels should be used. If you specify **Y** to bypass panel processing, then data model records must be used for these functions:

- T101 -- Obtain record create resource
- T102 -- Create record
- T103 -- Obtain record update resource
- T105 -- Update record
- T108 -- Obtain add record relation resource
- T109 -- Add record relation

If you specify to use bypass panel processing, then data model records can optionally be used for these functions:

- T100 -- Retrieve record
- T106 -- Obtain inquiry resources
- T107 -- Record inquiry

PICADMRC

Data model record indicator. This is set by your application. A **Y** indicates that the PIDT name (PICATABN) is a data view record ID. The data view record is used to build the PIDT.

It can take a significant amount of time to generate a PIDT from data view records. The length of time depends on the number of data attribute records (and validation

records that they reference) contained in the data view record. As with any PIDT, you can maintain the PIDT in storage for subsequent use.

Any value other than **Y** indicates that the PIDT name is a static or model PIDT to be used to find the table in the table data set or for dynamic processing.

PICADFMT

Date format indicator. This is set by your application. This enables the option of having all dates converted to or from a specified format, regardless of the database date format. Possible values are:

Value	Date format
X'00'	Database date format used. PICADSEP is ignored.
X'01'	MM/DD/YY MM-DD-YY MM.DD.YY
X'02'	MM/DD/YYYY MM-DD-YYYY MM.DD.YYYY
X'03'	DD/MM/YY DD-MM-YY DD.MM.YY
X'04'	DD/MM/YYYY DD-MM-YYYY DD.MM.YYYY
X'05'	YY/MM/DD YY-MM-DD YY.MM.DD
X'06'	YYYY/MM/DD YYYY-MM-DD YYYY.MM.DD
X'07'	DDMMMYY
X'08'	DDMMMYYYY
X'09'	YYDDD
X'0A'	YYYYDDD

PICADSEP

Date separator character indicator. For values in the range X'01' through X'06' for PICADFMT, in field PICADSEP you specify the character which will serve as the data separator; only the characters **slash** (/) or **hyphen** (-) or **period** (.) are valid.

PICACHAP

Change record approval status indicator. This is set by your application. An A indicates that the approval status for the change record is “Accept”. Any other value in this field indicates that the change record approval status is “Reject”.

PICARSV2

An 8-byte area reserved for future use. This area must be set to all binary zeros. Your application sets this field.

PICAUTSP

The name of a TSP or TSX to be invoked by the T111 transaction. A string of up to

255 characters can be passed to the TSP (in the variable data area) or TSX (as an argument) by storing the address of the string in PICAPARM and the length of the string in PICAPARL.

PICAPARL
If 0, then PICAPARM is the address of a user buffer; if greater than 0, then PICAPARM is the address of a string to be passed and PICAPARL is the length of that string.

PICALSTM
Used to specify how lists should be processed. Specify U to indicate that any new list data specified on the update will update existing lists in the record; specify A to indicate that any new list data specified on the update will be appended to the end of existing lists in the record; specify R to indicate that any new list data specified on the update will replace existing lists in the record. The default is U.

PICASAUD
Used to specify whether text audit data is to be retrieved. This field should be set to **Y** if text audit data suppression is requested. Any other value indicates that text audit data is to be returned.

PICARSV3
A 12-byte area reserved for future use. This area must be set to all binary zeros. Your application sets this field.

PICATZON
An 8–character field with the desired TIMEZONE label value (right-pad the field if the value is less than 8 characters). The value entered must match one of the values specified in the TIMEZONE record. Your application sets this field.

PICAAPVR
An 8-character field with the name of the privilege class that is approving or rejecting the change (right-pad the field if it is less than 8 characters). Your application sets this field.

PICATBFL
A 4-byte fixed field containing the total length of the data in the text argument buffer. If no text arguments exist, this value should be set to 0. Your application sets this field.

PICATBUF
A 4-byte pointer field containing the address of the text argument buffer. Your application sets this field.

PICARSV4
A 28-byte area reserved for future use. This area must be set to all binary zeros. Your application sets this field.

Program Interface Alias Table (PALT)

Alias tables let your applications:

- Specify alias names for PIDT member names so remote locations accessing the same database can identify a given PIDT by different alias names, or different PIDT versions using the same alias name.
- Specify an alias name for a p-word when building freeform search arguments. An example of a p-word is PERS/.

- Specify an alias name for a p-word index or s-word index for use in create record (T102), update record (T105), record inquiry (T107), and add record relation (T109) transactions. For example, you could use an alias name of "status" instead of s-word index S0BEE.
- Specify default response data values that can be used when an application does not provide a response value.

You create alias tables using the table build utility BLGUT8. PALTs are stored as members in partitioned data set BLGFMT. For more information about building alias tables, see "Field Validation Using the Field Validation Module BLGPPFVM" on page 279.

Table 32 shows the structure of the PALT and the page number where the table fields are explained.

Note: All character fields in the table are left justified and padded with blanks. As shown in the table, fields are set by the interface. Your application must not attempt to set these fields. If it does, results are unpredictable. In this case, *interface* can mean either the API or the table build utility.

Table 32. Alias Table (PALT)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
HEADER			**ALIAS TABLE HEADER 1st RECORD OF TABLE MEMBER**		
PALTACRO	0(0)	4	Acronym of PALT (character)	Interface	113
PALTBLNM	4(4)	8	Name of this table (character)	Interface	114
PALTRCDS	12(C)	4	Number of records per row (fixed) value = 2	Interface	114
PALTNUMR	16(10)	4	Number of rows in table (fixed)	Interface	114
PALTRSV1	20(14)	60	Reserved	--	114
ENTRY ROW			**ALIAS TABLE ROW 2nd AND SUBSEQUENT ROWS OF TABLE**		
PALTESLN	0(0)	4	Length of alias value (fixed)	Interface	114
PALTDDLN	4(4)	4	Length of default data (fixed)	Interface	114
PALTPDLN	8(8)	4	Length of p-word (fixed)	Interface	114
PALTSYMB	12(C)	5	Internal index symbol (character)	Interface	114
PALTESYM	17(11)	32	Alias value (character)	Interface	114
PALTDEFD	49(31)	45	Default data (character)	Interface	114
PALTPRFX	94(5E)	6	P-Word (character)	Interface	114
PALTTBLN	100(64)	8	PIDT member name (character)	Interface	114
PALTRSV2	108(6C)	52	Reserved	--	114

The following list describes the purpose of each field of a PALT header.

PALTACRO

A 4-character field containing the character string PALT to identify this program interface alias table. The table build utility sets this field.

PALTBLNM
An 8-character field containing the name of this table. The table build utility sets this field.

PALTRCDS
A 4-byte fixed field containing the number of records in each table row. The table build utility sets this field.

PALTNUMR
A 4-byte field containing the number of rows in the table structure. The table build utility sets this field.

PALTRSV1
A 60-byte area reserved for future use.

The following section describes the purpose of each field of a PALT row:

PALTESLN
A 4-byte fixed field containing the length of the alias value in PALTESYM. The table build utility sets this field.

PALTDDLN
A 4-byte fixed field containing the length of the default data in PALTDEFD. The table build utility sets this field.

PALTPDLN
A 4-byte fixed field containing the length of the p-word in PALTPRFX. The table build utility sets this field.

PALTSYMB
A 5-character field containing the internal symbol name (PIDTSYMB) of a PIDT row. The table build utility sets this field.

PALTESYM
A 32-character field containing the alias value name. The table build utility sets this field.

PALTDEFD
A 45-character field containing the response to be stored in the LLAPI response buffer. The table build utility sets this field.

PALTPRFX
A 6-character field containing the p-word used when constructing freeform arguments. The table build utility sets this field.

PALTTBLN
An 8-character field containing a left-justified, 1- to 8-byte PIDT member name. Currently only 1- to 7-byte member names are supported. The table build utility sets this field.

PALTRSV2
A 52-byte area reserved for future use.

For an example of a PALT, see “Program Interface Alias Table (PALT)” on page 112.

Program Interface Data Table (PIDT)

The program interface data table (PIDT) is a view of a particular type of Tivoli Information Management for z/OS database record. The PIDT is distinct from data view records and data

attribute records; but all —a PIDT, a data view record, and a data attribute record— identify fields within a Tivoli Information Management for z/OS record using Tivoli Information Management for z/OS's prefix word (p-word) and structured word (s-word) indexes, and panel names. On a record retrieve, you can provide a pre-defined view, or request that the LLAPI build a dynamic view based on the data contained in the record. You can tailor the pre-defined view (either with a static PIDT or data view record), or if using the LLAPI, request that the LLAPI build a dynamic view to meet the needs of your application and your customized Tivoli Information Management for z/OS database. When data model records are used, the data view record is used to generate the PIDT. In addition, Tivoli Information Management for z/OS supplies the Table Build Utility, BLGUT8, to assist you in creating these static PIDTs and models for dynamic PIDTs. The Table Build Utility BLGUT8 is described in *Tivoli Information Management for z/OS Operation and Maintenance Reference*. Tivoli Information Management for z/OS also contains some static PIDTs in the BLGFMT data set.

You can use data model records as a substitute for static PIDTs. Data view records define the fields that your application can access. Data attribute and validation records define the field attributes. If you use data model records, PIDTs are generated from these records to be used by your application and the API. "Field Validation Using the Field Validation Module BLGPPFVM" on page 279 contains additional information about data model records.

A data buffer is associated with the PIDT. This buffer holds response data associated with each field defined in the PIDT. The LLAPI obtains the storage for this buffer. You create static PIDTs by using the table build utility. The *Tivoli Information Management for z/OS Operation and Maintenance Reference* contains additional information about the table build utility. A PIDT created by BLGUT8 will contain a version number to indicate that the PIDT contains entries for the data type field. If your PIDT tables were created using the non-NLS Application Program Interface, you must recreate your tables using BLGUT8. If you have not migrated, processing will terminate.

There are three types of PIDTs:

Static PIDTs
: These are PIDTs built by BLGUT8 and stored in a partitioned data set that is a member of the report format table data set concatenation. Static PIDTs are further described in "Static PIDTs".

Dynamic PIDTs
: These are built from a retrieved record; they are further described in "Dynamic PIDTs" on page 116.

"Generated" PIDTs
: These are "generated" from a data view record and associated data attribute and validation records. A generated PIDT is used internally and is never actually written to a data set.

Static PIDTs

You can use the static PIDTs shipped with the licensed program for problem, change, and configuration records in the Tivoli Information Management for z/OS database, or you can define your own static PIDTs with the table build utility. Some reasons why you might want to define your own static PIDTs are:

- To represent a customized version of problem, change, or configuration records
- To represent user-defined record types

- To conserve storage and processing time by defining a PIDT customized to contain only the information needed by a particular application
- To use as a model for the record retrieve transaction (T100) when requesting a dynamic PIDT.

The API defines (as shipped PIDTs) a separate PIDT for each record process, such as inquiry, retrieve, create, update, and add record relation. PIDTs are stored as members in a partitioned data set called BLGFMT. The LLAPI uses these tables when it processes interface transactions.

Table 33 shows the structure of the PIDT and the page number where the table fields are explained.

Dynamic PIDTs

A dynamic PIDT is one whose header is defined by a model PIDT and whose entry fields and data buffers are defined by a record read from the Tivoli Information Management for z/OS database. The LLAPI can build a dynamic PIDT only on the record retrieve transaction (T100). The dynamic PIDT can then be used for a record create (T102) or update (T105) transaction.

The *model* PIDT for a dynamic PIDT can be any previously defined PIDT. It can be one that was generated dynamically by the record retrieve transaction (T100), one that was built by the table build utility for use with a record process, or one built by the table build utility specifically for use by the retrieve transaction in building dynamic PIDTs (built with USE (Header)).

Dynamic PIDTs cannot be used for Create (T102) or Update (T105) if bypass panel process (PICADRIF=Y) is specified at initialization. Dynamic PIDTs cannot be used for Retrieve (T100), Create (T102), or Update (T105) if data model records (PICADMRC=Y) are used.

Program Interface Data Table Fields

Each row of the PIDT represents either a visible or keyword phrase item, a response field, a direct add item, or text that is contained within a Tivoli Information Management for z/OS record. The API uses specific transactions to allocate a PIDT for its intended use.

Note: All character fields are left-justified and padded with blanks. As shown in the table, some fields are set by the interface. Your application should *not* attempt to set these fields. If it does, results are unpredictable. In this case, *interface* can mean either the API or the table build utility.

Some fields can be set by either the API or the application. For some of these fields, the application must only set them in a dynamic PIDT. Refer to the detailed descriptions of the fields for more information.

Table 33. Program Interface Data Table (PIDT)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
HEADER			**DATA TABLE HEADER 1st RECORD OF TABLE MEMBER**		
PIDTACRO	0(0)	4	Acronym of PIDT (character)	Interface	120
PIDTNAME	4(4)	8	Name of this table (character)	Interface	120

Table 33. Program Interface Data Table (PIDT) (continued)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
PIDTPTNM	12(C)	8	Associated PIPT name (character)	Interface	120
PIDTPIPT	20(14)	4	Associated PIPT pointer (pointer)	Interface	120
PIDTPIAT	24(18)	4	Associated PIAT pointer (pointer)	Interface	120
PIDTRCDS	28(1C)	4	Number of records per row (fixed) value=2	Interface	120
PIDTNUMR	32(20)	4	Number of rows in table (fixed)	Interface	120
PIDTBUFP	36(24)	4	Response buffer pointer (pointer)	Interface	120
PIDTBUFL	40(28)	4	Length of response buffer (fixed)	Interface	120
PIDTUSEF	44(2C)	1	Table usage field C=CREATE, R=RETRIEVE I=INQUIRY U=UPDATE A=ADD D=DYNAMIC H=HEADER (character)	Interface	121
PIDTSEPC	45(2D)	1	Response separator (character)	Interface	121
PIDTAUTH	46(2E)	2	Authorization code (character)	Interface	121
PIDTVERS	48(30)	1	PIDT version number (character)	Interface	121
PIDTDMRC	49(31)	1	Data model record indicator (Y or N) character	Interface	121
PIDTRSV1	50(32)	1	Reserved	--	121
PIDTDELO	51(33)	1	Delete entry types of Other (PIDTRDEF=O) indicator (character Y). Valid for dynamic PIDTs only.	Either	121
PIDTSPCP	52(34)	4	Pointer to free buffer space (pointer). Valid for dynamic PIDTs only.	Either	121
PIDTSPCE	56(38)	4	Pointer to end of free buffer space (pointer). Valid for dynamic PIDTs only.	Interface	121
PIDTPIHT	60(3C)	4	Associated PIHT pointer (pointer)	Either	122
PIDTRSV2	64(40)	16	Reserved	--	122
ENTRY ROW			**DATA TABLE ROW 2nd AND SUBSEQUENT ROWS OF TABLE**		
PIDTSYMB	0(0)	5	Field symbolic name (character)	Interface	122
PIDTRDEF	5(5)	1	Row definition field - R=Response, P=Phrase, D=Direct, X=Text, O=Other (character)	Interface	122
PIDTCODE	6(6)	2	Field error code (character).	Interface	123
PIDTMNCR	8(8)	4	Field's maximum number of entry responses (fixed).	Either	124
PIDTCNFR	12(C)	4	Number of field items (fixed).	Either	125
PIDTMAXL	16(10)	4	Field's maximum data length (fixed).	Either	125
PIDTCURL	20(14)	4	Field's current data length (fixed).	Either	125

Table 33. Program Interface Data Table (PIDT) (continued)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
PIDTDATP	24(18)	4	Response buffer data pointer (pointer).	Either	126
PIDTFPAT	28(1C)	4	PIPT row number of first pattern (fixed). Valid only for table rows where PIDTRDEF=R.	Interface	126
PIDTREQD	32(20)	1	Field defined as required (Y or N) (character). Valid for all table rows.	Interface	126
PIDTDATE	33(21)	1	Field defined as a DATE (Y or N) (character). Valid only for table rows where PIDTRDEF=R.	Either	126
PIDTSRCH	34(22)	1	Field defined as SEARCHABLE (Y, N or P=p-word only) (character). Valid only for table rows where PIDTRDEF=R, P, or D.	Either	126
PIDTJRNL	35(23)	1	Field defined as JOURNALED (F, O, N) (character). F = first, O = order, N = not journalized. Valid only for table rows where PIDTRDEF=R or D.	Either	126
PIDTLIST	36(24)	1	Field defined as a LIST ITEM (Y or N) (character). Valid only for table rows where PIDTRDEF=R.	Interface	126
PIDTRTYP	37(25)	1	Field defines record type (Y or N) (character). Valid only for table rows where PIDTRDEF=P.	Either	127
PIDTFAUP	38(26)	1	Field defined with authorization processing (Y or N) (character). Valid for all table rows.	Interface	127
PIDTSDAT	39(27)	1	Field defined as STRING DATA (Y or N) (character). Valid only for table rows where PIDTRDEF=R.	Interface	127
PIDTLZPD	40(28)	1	Field defined as LEFT ZERO PAD (Y or N) (character). Valid only for table rows where PIDTRDEF=R.	Interface	127
PIDTNOTL	41(29)	1	Field defined as use NOT LOGIC when collecting (Y or N) (character). Valid only for table rows where PIDTRDEF=R, P, or D.	Interface	127
PIDTPNLN	42(2A)	8	Panel name (character). Valid for all table rows.	Either	127
PIDTINDX	50(32)	2	Internal index (0000-FFFF) (internal form). Valid for all table rows.	Either	128
PIDTSWDD	52(34)	10	S-Word (internal form). Valid for all table rows.	Either	128
PIDTPFXD	62(3E)	6	P-Word (character). Valid only for all table rows where PIDTRDEF=R or D.	Either	128
PIDTPNLT	68(44)	1	Copied panel type field (fixed). Valid for all table rows.	Interface	128

Table 33. Program Interface Data Table (PIDT) (continued)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
PIDTMNTF	69(45)	1	Copied maintenance flag field (fixed). Valid for all table rows.	Interface	128
PIDTSWDL	70(46)	2	Length of s-word field PIDTSWDD (fixed). Valid for all table rows.	Either	128
PIDTPFXL	72(48)	4	Length of p-word field PIDTPFXD (fixed). Valid only for table rows where PIDTRDEF = R or D.	Either	128
PIDTGRPX	76(4C)	1	Group prefixing associated with this item (Y or N) (character). Valid only for table rows where PIDTRDEF = R.	Interface	128
PIDTREPL	77(4D)	1	Replace previous reply indicator (character Y). Valid for dynamic PIDTs only.	Either	129
PIDTFLAG	78(4E)	1	File processing indicator (character). Valid for dynamic PIDTs only.	Interface	129
PIDTCHNG	79(4F)	1	Change indicator (character). Valid for dynamic PIDTs only.	Either	129
PIDTVISL	80(50)	2	Length of visible phrase (fixed). Valid only for table rows where PIDTRDEF=P or D.	Interface	129
PIDTVISD	82(52)	28	Visible phrase data (character). Valid only for table rows where PIDTRDEF=P or D.	Interface	129
PIDTDTYP	110(6E)	1	Data type field (M=Mixed, S=SBCS, D=DBCS) Valid only for table rows where PIDTRDEF=R.	Interface	129
PIDTDIAG	111(6F)	1	Dialog indicator for record (character). Valid for dynamic PIDTs only.	Either	129
PIDTVLDD	112(70)	10	Original s-word, prefix, or panel name (character). Valid for dynamic PIDTs only.	Interface	130
PIDTVLDL	122(7A)	2	Length of value stored in PIDTVLDD (fixed). Valid for dynamic PIDTs only.	Interface	130
PIDTVREC	124(7C)	8	Validation record ID	Interface	130
PIDTVSWD	132(84)	10	S-word of the validation record ID	Interface	130
PIDTDSWD	142(8E)	10	Root s-word in validation record for the validation data list.	Interface	130
PIDTVALE	152(98)	1	If this value is **Y**, indicates that the PIDT entry is validated when the entry has data and is processed by the API.	Either	130
PIDTCSVL	153(99)	1	Exact case validation flag (Y or N) (character). Valid only for table rows where PIDTRDEF=R.	Interface	130
PIDTCGMX	154(9A)	1	Mixed case cognizing flag (Y or N) (character). Valid only for table rows where PIDTRDEF=R.	Interface	130

Table 33. Program Interface Data Table (PIDT) (continued)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
PIDTCDCA	155(9B)	1	Case conversion indicator. Valid values are: U=all upper case L=all lower case F=first character upper case; remainder lower P=case as defined in literal validation pattern A=as entered by the application Any other value is treated as U Valid only for table rows where PIDTRDEF=R.	Interface	130
PIDTVPAT	156(9C)	4	Copy of PIDTFPAT information.	Interface	131

The following section describes the purpose of each field of a PIDT header:

PIDTACRO
A 4-character field containing the character string PIDT to identify this program interface data table. The interface sets this field.

PIDTNAME
An 8-byte character field containing the name of this PIDT or the record ID of a data view record. The interface sets this field. For a dynamic PIDT, the LLAPI adds an * to the name.

PIDTPTNM
An 8-byte character field containing the associated PIPT name. The interface sets this field.

PIDTPIPT
A 4-byte pointer to the associated PIPT. The LLAPI sets this field.

PIDTPIAT
A 4-byte pointer to the associated PIAT. You use this field only for inquiry processing that includes freeform search arguments. The LLAPI sets this field.

PIDTRCDS
A 4-byte fixed field containing the number of records per row. The interface sets this field.

PIDTNUMR
A 4-byte fixed field containing the number of rows in this table. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field.

PIDTBUFP
A 4-byte pointer to the data response buffer. The API sets this field.

PIDTBUFL
A 4-byte fixed field containing the data response buffer length. The LLAPI sets this field.

PIDTUSEF

A 1-byte character field defining the functional use of the table. The interface sets one of the following values in this field:

- C for record create
- R for record retrieval
- I for record inquiry
- U for record update
- A for add record relation
- H for header-only

For a dynamic PIDT, the LLAPI sets the following value in this field:

- D for dynamic

PIDTSEPC

A 1-byte character field containing the response separator character. The response separator character separates multiple response items in the response buffer or indicates to delete the item. The interface sets this field.

PIDTAUTH

A 2-byte character field containing the authorization code used to verify the transaction. The interface sets this field.

PIDTVERS

A 1-byte character field containing the version number of the PIDT used to verify the PIDT has been migrated to an NLS version or version 6.1. The interface sets this field.

PIDTDMRC

A 1-byte character field containing a **Y** or **N**. When this field is **Y**, it indicates that PIDTNAME contains a data view record ID. When this field is **N**, it indicates that PIDTNAME contains the name of a PIDT. This is set by the interface.

PIDTRSV1

A 1-byte area reserved for future use.

PIDTDELO

A 1-byte character field indicating whether entry types of Other (PIDTRDEF=O) are to be removed or excluded from the record updated or created by this PIDT. A value of **Y** in this field indicates entry types of other are to be excluded. If any other character is in this field, entry types of other are not excluded but are processed on a one-to-one basis. This field is valid for dynamic PIDTs only. The LLAPI sets this field to **N**. The application can update it.

PIDTSPCP

A 4-byte pointer to the free space in the data buffer that was added as a result of the application providing a value in the PICAREQL field on a record retrieve transaction (T100) that uses a dynamic PIDT. If the application does not ask for extra space, then the value is set to the end of the last response of the last PIDT entry. This field is valid for dynamic PIDTs only. The LLAPI sets this field and the application can update it.

PIDTSPCE

A 4-byte pointer to the end of the free space in the data buffer that was added as a result of the application providing a value in the PICAREQL field on a record retrieve transaction (T100) that uses a dynamic PIDT. When generating a dynamic PIDT:

- If PIDTMAXL plus PIDTSPCP is less than or equal to PIDTSPCE, there is enough space to move the entry to the buffer pointed to by PIDTSPCP for a create (T102) or update (T105) transaction.
- If PIDTMAXL plus PIDTSPCP is greater than PIDTSPCE, there is not enough space and your application needs to do another record retrieve transaction asking for more free space in the buffer.

This field is valid for dynamic PIDTs only. The LLAPI sets this field.

PIDTPIHT
A 4-byte pointer to the associated PIHT. Your application or the LLAPI sets this field.

PIDTRSV2
A 16-byte area reserved for future use.

The following section describes the purpose of each field of a PIDT row:

PIDTSYMB
A 5-byte character field containing the symbolic name given to the record type, data, or text attribute associated with this table row. This is the character representation of a s-word index or p-word index or the character string Xnnnn if retrieving freeform text with a dynamic PIDT where nnnn starts at 0001 and increases with each freeform text item in the unique text record. It defines a record type, visible phrase, data response, or text item. If the data type is defined with an s-word, the character S precedes the s-word index. If the attribute is defined with a p-word, the character P precedes the p-word index. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field and the application can update it.

PIDTRDEF
A 1-byte character field containing the definition of the kind of data that is collected by this row. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field and the application can update it.

- When this field contains the character R, it signifies that this row collects a response field. The Tivoli Information Management for z/OS problem reporter name collected from panel BLG6REQN is an example of a response field.
- When this field contains the character P, it signifies that this row collects an s-word and visible phrase. The problem record type (TYPE=PROBLEM) and its associated s-word is an example of this entry.
- When this field contains the character D, it signifies that this row defines a direct add item. When processing this field interactively, control panels collect direct add items using the data specified in ADD control lines. When the LLAPI adds this data, validation is not performed. Therefore, there is no corresponding PIPT entry for this field. Panel BLG1A111 is an example of a panel that collects direct add fields.
- When this field contains the character X, it signifies that this table row identifies a text field. Tivoli Information Management for z/OS problem description text is an example of a text field.
- When this field contains the character O, it signifies that the SDE for this row does not contain an s-word or a p-word.

PIDTCODE

A 2-byte character field containing the error code associated with missing or incorrect PIDT field specifications. The LLAPI sets this field.

The API returns one of the following codes:

00 No error detected.

01 A response field has a length greater than PIDTMAXL. The LLAPI processes this data, but truncates the collected data to the length specified in PIDTMAXL.

02 Field PIDTDATP contains a valid pointer, but either or both PIDTCURL and PIDTCNFR contain a zero value.

03 The number of responses found does not match the value in PIDTCNFR.

04 A required field has no response data. Field PIDTREQD=Y and PIDTDATP, or PIDTCNFR, or PIDTCURL contain a zero value.

05 TABLE/RECORD string data response conflict. Your application specified a response entry as string data (PIDTSTAT=Y) in the table entry but the response contained in the record is not string data. Or, your application's response entry was not string data (PIDTSTAT=N) in the table entry but the response contained in the record is string data. This code can only be set by retrieve processing.

06 Record ID format error. This field must be 8 numeric characters (but *not* all zeroes), or the first character must be A-Z with remaining characters being A-Z, 0-9, /, #, $, @, or & with no imbedded blanks.

07 The text data set name specified for a text row is greater than 44 characters.

08 The size of the response data specified is larger than can be collected in a Structured Data-Entry (SDE).

This could occur when entering a nonlist item for which there are multiple responses and there is too much data for the SDE to hold.

09 Text processing errors occurred. This error applies only to rows where PIDTRDEF contains X.

10 The number of responses specified is more than the number allowed. Field PIDTCNFR is greater than PIDTMNCR.

11 Text data set allocation error. This error applies only to rows where PIDTRDEF contains X.

12 Date field is not valid. This error only applies to rows where PIDTRDEF contains the character R and PIDTDATE contains Y.

14 Text unit specification is not valid because:

- When PICATXAU is not **Y**, PIDTCURL divided by PIDTCNFR results in a value greater than 132, or produces a remainder, or both.
- When PICATXAU is **Y**, PIDTCURL divided by PIDTCNFR results in a value greater than 168 or less than 37, or produces a remainder, or both.

15 Field contains incorrect mixed data.

16 Field data type does not match the data type value specified by PIDTDTYP.

17 Data type field PIDTDTYP contains an incorrect value.

33 The PIDTSYMB value for this PIDT entry is not unique. A previous PIDT entry has the same value.

34 The p-word in the PIDT entry does not contain / or _ in the first 6 positions.

35 When doing an update with a dynamic PIDT, the one-to-one correspondence of PIDT entry to record entry was not maintained.

36 When doing an update with a dynamic PIDT, the PIDT entry has an s-word length greater than 10 or a p-word length greater than 6.

37 Unable to locate the data pointed to by PIDTDATP. It may not be valid DBCS data.

38 When doing an update with a dynamic PIDT, a list has multiple responses for a list item.

39 When doing an update with a dynamic PIDT, the s-word suffix of a list item is less than the suffix for a preceding list item for that same list.

40 Equal sign processing error. The data required to complete the automatic entry could not be located.

41 A program exit encountered an error while processing the PIDT entry.

42 The entry encountered an error attempting to access the validation record from the data model database. The record ID is specified in the PIDTVREC field.

43 No validation patterns were found in the attribute or validation record, or else the ID of the validation record cannot be found in the data of the current record. Finding the record ID to use for validation data depends on whether an s-word is specified in PIDTVSWD. If an s-word is specified in PIDTVSWD, the current record is searched for the s-word and the data associated with it is used as the validation record ID. If PIDTVSWD is not specified, or the record ID cannot be found, then the record ID specified in PIDTVREC is used if it exists. The database that contains the data model records is specified in the session parameters (DMODELDB=), or a default of database 5 is used.

44 The validation s-word specified is too long. The s-word must be eight or fewer characters. Validation data is stored as list data and must be eight characters or fewer. The entry cannot be processed.

45 The entry encountered an error attempting to access the validation record from the data model database. The record ID used was found by searching the current record for the s-word specified in PIDTVSWD.

46 The entry encountered an error attempting to access the data attribute record from the data model database. The record ID is specified in the PIDTVREC field.

PIDTMNCR

A 4-byte fixed field containing the maximum number of responses that the API collects for a field at one time in create mode. Create mode is defined as collecting responses when either creating or updating a Tivoli Information Management for z/OS record. This field provides the application with information on how this field is defined in panel dialogs so the field can be simulated in the LLAPI. For replaceable,

table-list entry fields, and string data fields, this value is usually set to 1. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field, and your application can update it. For entry types of Other (PIDTRDEF=O) in a dynamic PIDT, your application must not change this field because the field contains the response number obtained from the SDE when retrieved.

The interface uses the field PIDTMAXL to calculate and set this field if the assisted-entry panel field REPLY VALUE MEANING is set to LENGTH. The calculation is based on the maximum allowable number of 1-character responses.

PIDTCNFR

A 4-byte fixed field. When you use this table for record retrieval, this field's value represents the number of responses or text units (lines) found for this field. When you use this table for record entry or inquiry, this value represents the number of responses or text units (lines) stored in the response buffer for this field or the number of times a phrase or direct data item is collected.

When this field has a value of zero and PIDTRDEF=P, D, or O, the entry is excluded from the record on a create transaction (T102).

You delineate each entry for a field in the buffer with a separator character unless the field is a string field. The value specified in this field must not exceed the value specified in PIDTMNCR for table rows not having PIDTLIST=Y. The API or the application sets this field. Validation of this field is carried out by comparing it with the actual number of responses counted in the field response data. The calculation of the actual number of responses is based on the number of response separator characters, as specified by PIDTSEPC, that occur in the data. The separator characters will only be located in single-byte character set (SBCS) portions of the data.

PIDTMAXL

A 4-byte fixed field containing the maximum length of a response or text data set name. Where the interface collects multiple responses, this is the maximum size of a single response. This field is nonzero only for table rows containing the characters R or X in field PIDTRDEF.

The interface uses the field PIDTMNCR to calculate and set this field if the assisted-entry panel field REPLY VALUE MEANING is set to WORDS. The calculation is based on the validation pattern. For a dynamic PIDT, the LLAPI sets this field and your application can update it. When set by a record retrieve using dynamic record retrieval, this value is based on the size of the data in the record plus the value specified in PICAESPC. The value does not correspond to the actual maximum length of the field defined in your panels.

If this API connects to a BLX-SP that supports DBCS (that is, DBCS=YES is specified in the BLX-SP parameters), the maximum value of this field is 32 767.

PIDTCURL

A 4-byte fixed field containing the length of the field response data or text data set name that is currently stored in the response buffer. If the API collects list entry responses, this length is the total length of all responses, including response separators. This field is nonzero only for table rows containing the characters R or X in field PIDTRDEF. Your application or the LLAPI sets this field.

If this API connects to a BLX-SP that supports DBCS (that is, DBCS=YES is specified in the BLX-SP parameters), the maximum value of this field is 32 767.

PIDTDATP

A 4-byte pointer to the field response data or text data set name that is stored in the response buffer. This field is nonzero only for table rows that contain the character R or X in field PIDTRDEF. This address must be within the range of addresses set by PIDTBUFP through (PIDTBUFP + PIDTBUFL - 1). This field is set by either the API or the application.

If this API connects to a BLX-SP that supports DBCS (that is, DBCS=YES is specified in the BLX-SP parameters), the maximum length of the data pointed to by this field is 32 767.

PIDTFPAT

A 4-byte fixed field containing the first pattern row number in the PIPT that corresponds to this field. This field is nonzero only for table rows that contain the character R in field PIDTRDEF. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field for entries that have multiple p-words.

PIDTREQD

A 1-byte character field containing a **Y** or an N. When this field is Y, it indicates that the API must collect this field to complete the transaction. When this field is N, it indicates that the field is not required. This field is valid for all table rows. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field to N for all entries.

PIDTDATE

A 1-byte character field containing a **Y** or an N. When this field is Y, it indicates that this response field collects date responses. When this field is N, it indicates that this is not a date field. The API enters date response fields in their external date format then converts them using the external date conversion routine. This field is valid only for table rows that contain the character R in field PIDTRDEF. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field and the application can update it.

PIDTSRCH

A 1-byte character field containing a searchable field indicator of Y, N, S or P. When this field is Y, it indicates that this response field is cognized and can be used when building inquiry arguments. When this field is N, the field is defined as not searchable. When this field is S, it indicates that only the s-word is cognized. When this field is P, it indicates that the s-word is not cognized, and the p-word is cognized. This field is valid only for table rows that contain the character R, P, or D in field PIDTRDEF. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field and the application can update it.

PIDTJRNL

A 1-byte character field containing a journal field indicator. When this field contains an F or O, it indicates journalization for this response field. F indicates journalized first, and O indicates journalized in occurrence order. N indicates the field is not journalized. This field is valid only for table rows that contain the character R or D in field PIDTRDEF. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field and the application can update it.

PIDTLIST

A 1-byte character field containing the table-list item field indicator. When this field is **Y** it indicates that this response field is defined as a table-list item. When this field is N, the API does not perform table-list item processing. The interface

processes and collects table-list items using dynamic s-words that allow record entry of multiple entries of the same data type. This field is valid only for table rows that contain the character R in field PIDTRDEF. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field.

PIDTRTYP

A 1-byte character field containing a record type indicator. When this field is Y, it indicates that this table entry defines the record type. When this field is N, it indicates that this field does not define a record type. This field is valid only for table rows that contain the character P in field PIDTRDEF. A row of this type is required for all tables except inquiry tables. The interface sets this field during table create processing. For a dynamic PIDT, the LLAPI or the application sets this field.

PIDTFAUP

A 1-byte character field containing a field authorization indicator. When this field is Y, it indicates that field authorization processing occurs for the field. When this field is N, it indicates that no field authorization occurs. The API does not perform field level authorization. This PIDT field lets your application perform field level authorization if required. This field is valid for all table rows and is set by the interface. For a dynamic PIDT, the LLAPI sets this field.

PIDTSDAT

A 1-byte character field containing a string data field indicator. When this field is Y, it indicates that this data attribute is defined as a string data field. When this field is N, it indicates that this is not a string data field. A string data field is treated as one response when field PIDTLIST contains the character N. The response can contain multiple words and special characters. If it is cognized, each word in the response is cognized separately without a p-word. When field PIDTLIST contains the character **Y** and field PIDTSDAT also contains the character Y, the string data in the response buffer is treated as multiple list responses. In this case, any separator character found in the response buffer is not considered as a special character that is part of a string data response, but as an indication of the end of a string data response. The separator character is defined in PIDTSEPC. The Tivoli Information Management for z/OS problem description abstract field is a string data field. This field is valid only for table rows that contain the character R in field PIDTRDEF. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field.

PIDTLZPD

A 1-byte character field containing the left zero-padded field indicator. When this field is Y, it indicates that this data attribute is left zero-padded to its maximum size. When this field is N, it indicates that no left zero padding is required. This field is valid only for table rows that contain the character R in field PIDTRDEF. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field to N.

PIDTNOTL

A 1-byte character field containing the use-not-logic indicator. When this field is Y, the API collects this data attribute using use-not-logic indicators. When this field is N, it indicates that not-logic collection is not performed. You use this field only when performing inquiry processing. This field is valid only for table rows that contain the character R, P, or D in field PIDTRDEF. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field to N.

PIDTPNLN

An 8-byte character field containing the panel name where the symbolic item is defined. For other types, this is the panel from which the selection was chosen. For

response items, this is an assisted-entry panel. For phrase items, this is the panel containing the phrase. For direct entry items, this is the control panel containing the ADD item control line. For text items, this is the name of the panel invoking the text processing program exit. The interface sets this field when it creates tables. For a dynamic PIDT, the LLAPI sets this field and the application can update it.

PIDTINDX
A 2-byte character field containing the internal form of the s-word index or p-word index (0000-FFFF). This field is valid for all table rows. It contains an s-word index if you defined the field using an s-word index. It contains a p-word index if you defined the field using a p-word index. It contains an s-word index if you defined the field using both an s-word index and a p-word index. It contains the field response number of the panel if this is a row in a dynamic PIDT with an entry type of Other (PIDTRDEF=O). The interface sets this field. For a dynamic PIDT, the LLAPI sets this field and the application can update it.

PIDTSWDD
A 10-byte character field containing the internal form of the s-word data for the response field, text item, or visible phrase. This field is valid for all table rows. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field and the application can update it.

PIDTPFXD
A 6-byte character field containing the p-word for the response or direct entry field. You can use this field to form inquiry arguments in the PIAT. This field is valid only for table rows that contain the character R or D in field PIDTRDEF. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field, and the application can update it.

PIDTPNLT
A 1-byte character field containing the panel type field data copied from the source panel. This field is valid for all table rows. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field.

PIDTMNTF
A 1-byte character field containing the maintenance field data copied from the source panel. This field is valid for all table rows. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field.

PIDTSWDL
A 2-byte fixed field containing the length of the s-word data at field PIDTSWDD. This field is valid for all table rows. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field and the application can update it.

PIDTPFXL
A 4-byte fixed field containing the length of the p-word at field PIDTPFXD. This field is valid for table rows containing R, P, or D in field PIDTRDEF. The interface sets this field. For a dynamic PIDT, the LLAPI sets this field, and the application can update it.

PIDTGRPX
A 1-byte character field indicating group prefixing. Group prefixing allows the API to collect multiple p-words for a response. Refer to the *Tivoli Information Management for z/OS Panel Modification Facility Guide* for more information.

This field is valid for table rows containing R in field PIDTRDEF. This field is set to **Y** when the assisted-entry panel validation control lines used to construct this table entry meet the following criteria:

- Begin with a multiple pattern group [1]
- Contain a multiple pattern group preceded by a nonmultiple pattern that begins with an equal character (=).

If a multiple pattern group is preceded by a nonmultiple pattern that does not begin with an equal character, the interface sets this field to **N** . The interface sets this field. For a dynamic PIDT, the LLAPI sets this field.

PIDTREPL

A 1-byte character field indicating whether the SDE built for this PIDT entry is to be marked replaceable. A value of **Y** indicates that when an SDE is built for this PIDT entry, it is marked replaceable. Any other value indicates that the SDE is marked non-replaceable. This field is valid for dynamic PIDTs only. The LLAPI sets this field, and the application can update it.

PIDTFLAG

A 1-byte character field reserved for dynamic PIDT processing. The LLAPI sets this field.

PIDTCHNG

A 1-byte character field for use on update transactions to indicate that the PIDT entry is to be processed. A value of **Y** in this field indicates that the record's data is to be updated with the dynamic PIDT's entry. If any other value is in this field, the entry is not updated. This field is valid for dynamic PIDTs only. The LLAPI sets this field to **N**. Your application can update it.

PIDTVISL

A 2-byte fixed field containing the length of the visible phrase at field PIDTVISD. This field is valid for table rows containing P or D in field PIDTRDEF. The API and the interface set this field. For a dynamic PIDT, the LLAPI sets this field, and the application can update it. This field is zero if PIDTVISD does not contain a visible phrase.

PIDTVISD

A 28-byte character field containing the visible phrase or direct add data, when collected. The maximum phrase length is 28 bytes. Data longer than 28 bytes is truncated. This field is valid only for table rows containing P or D in field PIDTRDEF. The LLAPI and the interface set this field. For a dynamic PIDT, the LLAPI sets this field, and the application can update it. If this field does not contain a visible phrase, the field PIDTVISL is zero.

PIDTDTYP

A 1-byte character field containing the data type. This field validates the type of data entered. It can contain a value of mixed (M), SBCS (S) or DBCS (D). This field is valid only for table rows that contain the character R in field PIDTRDEF. The interface sets this field.

PIDTDIAG

A 1-byte character field that holds any record dialog flags associated with this entry. A value of B in this field indicates the beginning of a dialog. A value of E in this

1. A group of prefixes that can be stored with a response

field indicates the end of a dialog. All other values are ignored. This field is valid with dynamic PIDTs only. The LLAPI sets this field, and the application can update it.

PIDTVLDD
A 10-byte character field containing either the PIDT s-word, p-word, or panel name. This field is valid for dynamic PIDTs only. The LLAPI sets this field.

PIDTVLDL
A 2-byte fixed field containing the length of the value stored in the PIDTVLDD field. This field is valid for dynamic PIDTs only. The LLAPI sets this field.

PIDTVREC
An 8-byte field which is the validation record ID found in the assisted-entry panel if the PIDT is built by the table build utility, or found in the data attribute record if the PIDT is built with a data view record.

PIDTVSWD
The s-word that identifies the validation record ID. It is found in the assisted-entry panel if the PIDT is built by the interface, or found in the data attribute record if the PIDT is built with a data view record.

PIDTDSWD
The root s-word of the validation data in the validation record. It is found in the assisted-entry panel if the PIDT is built by the interface, or found in the data attribute record if the PIDT is built with a data view record.

PIDTVALE
A 1-byte character field containing a validation indicator. A Y indicates that the PIDT entry should be validated when the entry has data and is processed by the API. The application should turn on this flag if equal sign processing is requested and the response data includes an equal sign. BLGPPFVM turns on this flag if there is a validation record ID s-word in the entry being validated so that validation will occur when the data is processed by a create, update, add record relation, or inquiry transaction. A value other than **Y** indicates no validation is needed. The LLAPI sets the field to **N** after validation occurs.

PIDTCSVL
A 1-byte flag which indicates whether the data is validated to ensure that the case of any letters matches those in the pattern. A **Y** indicates that the input data must match exactly any literal validations contained in the validation pattern, including the case of the data. Data validation only occurs if you call BLGPPFVM. If this value is *not* **Y**, the case of the input data is ignored when attempting to match a validation pattern. The API sets this field.

PIDTCGMX
If this value is **Y**, the search index entries for this field will be stored in mixed case and a case-sensitive search will be required to find them. The API sets this field.

PIDTCDCA
Defines the case in which the data should be stored in the record if validation is performed. Valid values are:

U=all upper case
L=all lower case
F=first character upper case; remainder lower
P=case as defined in literal validation pattern

A=as entered by the application
Any other value is treated as U

The input data is converted as indicated if you call BLGPPFVM to validate the data. The API sets this field.

PIDTVPAT

A 4-byte fixed field which replicates the PIDTFPAT information. This field keeps track of a PIPT stub entry used to build the actual PIPT entries when validation record IDs or validation record ID s-words are specified.

PIDT Example

Table 35 shows an example of PIDT rows with the values of significant entry fields indicated. Table 34 shows the PIDT header for this particular PIDT. For information about the PIDT header, see Table 33 on page 116. The example shows the PIDT for a problem create record, assuming that the PIDT has already been obtained with a PICAREQL of 50. The example shows the PIDT after data has been filled in for rows S0B59, S0BEE, and S0E0F.

Table 34. PIDT Example, Header Field Values

ACRO	NAME	PTNM	PIPT	PIAT	RCDS	NUMR	BUFP	BUFL	USEF	SEPC	AUTH
PIDT	BLGYPRC	BLGYPRCP	3000	0	2	7	2000	32	C	,	110

Table 35. PIDT Example, Entry Field Values

PIDT SYMBOL DESCRIPTION	PIDT RDEF	PIDT MNCR	PIDT CNFR	PIDT MAXL	PIDT CURL	PIDT DATP	SELECTED PIDT FIELD NAMES
S0032 Problem Record	P (Phrase)	1	1	0	0	0	SRCH=Y RTYP=Y FAUP=Y NOTL=N
S0B59 Reported by	R (Resp)	1	1	F	8	2000	REQD=Y RTYP=N DATE=N FAUP=N SRCH=Y SDAT=N JRNL=N LZPD=N LIST=N NOTL=N
S0C09 Problem Type	R (Resp)	1	0	8	0	0	REQD=N RTYP=N DATE=N FAUP=N SRCH=Y SDAT=N JRNL=O LZPD=N LIST=N NOTL=N
S0BEE Status	R (Resp)	1	1	7	4	2008	REQD=Y RTYP=N DATE=N FAUP=Y SRCH=Y SDAT=N JRNL=O LZPD=N LIST=N NOTL=N
S0C3D Date Occurred	R (Resp)	1	0	8	0	0	REQD=N RTYP=N DATE=Y FAUP=N SRCH=Y SDAT=N JRNL=N LZPD=N LIST=N NOTL=N

Table 35. PIDT Example, Entry Field Values (continued)

PIDT SYMBOL DESCRIPTION	PIDT RDEF	PIDT MNCR	PIDT CNFR	PIDT MAXL	PIDT CURL	PIDT DATP	SELECTED PIDT FIELD NAMES
S0E0F Description Abstract	R (Resp)	1	1	2D	15	200C	REQD=Y RTYP=N DATE=N FAUP=N SRCH=Y SDAT=U JRNL=N LZPD=N LIST=N NOTL=N
S0E01 Description Text	X (Text)	1	0	2C	0	0	FAUP=N JRNL=N SRCH=Y

In the PIDT buffer for this example would be a string of data that started at storage address 2000 and is 50 characters long.

```
'DOE/JOHNOPENTHIS IS A BAD PROBLEM                '
```

Program Interface History Table (PIHT)

The program interface history table (PIHT) contains history data. The API allocates storage and constructs a PIHT when your application requests history data processing on the LLAPI retrieve record (T100) transaction by setting the PICAHIST flag field to Y. When history data is retrieved for a record, the API stores the address of the PIHT in the PIDTPIHT field. The PIHT is freed when your application calls a free PIDT transaction (T006).

The PIHT consists of a header portion and a series of rows, where each row describes a piece of data. A history entry is composed of one or more rows grouped in sequence. Entries created by Information/Management Version 1 (PIHTVER1=Y) have only one row of data per group.

Note: The Version 1 indicated by PIHTVER1=Y means Version 1.0 of Information/Management, introduced in about 1980.

All other entries (PIHTVER1≠Y) can have one or more rows forming a group. When multiple rows are present, those with control data (PIHTCNTL=Y) must appear before those with regular data (PIHTCNTL≠Y). Control data was journalized with the specification of FIRST. Regular history data was journalized with ORDER specified.

Table 36 shows the structure of the PIHT and the page number where the table fields are explained.

Note: As shown in the table, fields are set by the interface. Some fields that are set by the interface can be updated by your application. Fields that can be updated by your application are indicated as set by *either*. If your application attempts to set a field that is indicated as set by the interface, results are unpredictable. In this case, *interface* means the API.

Table 36. History Table (PIHT)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
HEADER			**HISTORY TABLE HEADER 1st RECORD OF TABLE MEMBER**		
PIHTACRO	0(0)	4	Acronym of PIHT (character)	Interface	133
PIHTNUMR	4(4)	4	Number of rows in table (fixed)	Interface	133

Table 36. History Table (PIHT) (continued)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
PIHTRSV1	8(8)	72	Reserved	--	134
ENTRY ROW			**HISTORY TABLE ROW 2nd AND SUBSEQUENT ROWS OF TABLE**		
PIHTCODE	0(0)	2	Row error code (character)	Interface	134
PIHTVER1	2(2)	1	Version 1 history row indicator (character Y) **Note:** The Version 1 indicated by PIHTVER1=Y means Version 1.0 of Information/Management, introduced in about 1980.	Either	134
PIHTV1JD	3(3)	5	Julian date in YYDDD format (character). This field is valid only where PIHTVER1 = Y.	Either	134
PIHTSGRP	8(8)	1	Start of history group indicator (character Y). This field is valid only where PIHTVER1≠Y.	Either	134
PIHTCNTL	9(9)	1	Row contains control data indicator (character Y). This field is valid only where PIHTVER1≠Y.	Either	134
PIHTPWP	10(A)	1	History data is present indicator (character Y). This field is valid only where PIHTVER1≠Y.	Interface	135
PIHTLIST	11(B)	1	Data built by list processor indicator (character Y). This field is valid only where PIHTVER1≠Y.	Either	135
PIHTSWDL	12(C)	2	S-Word length (fixed). This field is valid only where PIHTVER1≠Y.	Either	135
PIHTSWDD	14(E)	10	S-Word for data (optional) (character). This field is valid only where PIHTVER1≠Y.	Either	135
PIHTRSV2	24(18)	4	Reserved	--	135
PIHTMAXL	28(1C)	2	Maximum data length (fixed)	Interface	135
PIHTCURL	30(1E)	2	Current data length (fixed)	Either	134
PIHTDATA	32(20)	32	History data field (character)	Either	135
PIHTRSV3	64(40)	16	Reserved	--	135

The following section describes the purpose of each field of a PIHT header:

PIHTACRO

A 4-character field containing the string PIHT to identify this table. The LLAPI sets this field.

PIHTNUMR

A 4-byte fixed field containing the number of rows in this table. The LLAPI sets this field.

PIHTRSV1
A 72-byte area reserved for future use.

The following section describes the purpose of each field of a PIHT row:

PIHTCODE
A 2-byte character field containing the error code associated with PIHT field specifications that are not valid. The LLAPI sets this field.

The API returns one of the following codes:

00 No error detected.

01 This row is the start of a multiple row group, but it does not have PIHTSGRP=Y specified.

02 This row contains control data (PIHTCNTL=Y), but it follows a row with no control data (PIHTCNTL≠Y) and is within the same multiple row group.

03 The current data length (PIHTCURL) for this row is larger than the maximum allowed value (PIHTMAXL).

04 The s-word data length (PIHTSWDL) for this row is larger than 10.

05 Reserved

06 The Julian date field (PIHTV1JD) for this row does not contain a valid date.

PIHTVER1
A 1-byte character field indicating whether the row contains history data created by Information/Management Version 1. A **Y** indicates the history data was created by Information/Management Version 1. Any other value indicates that the history data was created by Information/Management Version 2 or later. The LLAPI sets this field, and your application can update it.

Note: The Version 1 indicated by PIHTVER1=Y means Version 1.0 of Information/Management, introduced in about 1980.

PIHTV1JD
A 5-byte character field containing the Julian date of this history entry. The date is in the form YYDDD, where YY is the last two digits of the year and DDD is the number of the day in the year. This field is valid only for table rows that contain the character **Y** in field PIHTVER1. The LLAPI sets this field, and your application can update it.

PIHTSGRP
A 1-byte character field indicating whether this row starts a group of rows that comprise a history entry. A **Y** indicates the beginning of a history entry. Any other value indicates that this row is not the beginning of a history entry. This field is valid only for table rows that do not contain the character **Y** in field PIHTVER1. The LLAPI sets this field, and your application can update it.

PIHTCNTL
A 1-byte character field indicating whether this row contains history control data (data journalized with the specification of FIRST). A **Y** indicates that the row does contain history control data. Any other value indicates that this row contains data journalized with the specification of ORDER. This field is important for controlling how the data appears when History is selected from a Tivoli Information Management for z/OS panel or when a report is run. Within a history entry, all rows

that contain a **Y** in this field must come before the rows that do not contain a **Y**. This field is valid only for table rows that do not contain the character **Y** in field PIHTVER1. The LLAPI sets this field, and your application can update it.

PIHTPWP
A 1-byte character field indicating that history data is present. A Y indicates that history data is present. Any other value indicates that history data is not present. This field is valid only for table rows that do not contain the character **Y** in field PIHTVER1. The LLAPI sets this field.

PIHTLIST
A 1-byte character field indicating whether this history data was created by the list processor. A **Y** indicates that this history data was created by the list processor. Any other value indicates that this history data was not created by the list processor. This field is valid only for table rows that do not contain the character **Y** in field PIHTVER1. The LLAPI sets this field, and your application can update it.

PIHTSWDL
A 2-byte fixed field containing the length of the s-word data in field PIHTSWDD. A value of zero indicates that there is no s-word data. This field is valid only for table rows that do not contain the character Y in field PIHTVER1. The LLAPI sets this field. Your application must update this field if it changes the length of the s-word for the history data.

PIHTSWDD
A 10-byte character field containing the internal form of the s-word data for the history data. This field is valid only for table rows that do not contain the character **Y** in field PIHTVER1. The LLAPI sets this field, and your application can update this field if it changes the s-word for the history data.

PIHTRSV2
A 4-byte area reserved for future use.

PIHTMAXL
A 2-byte fixed field containing the maximum length of history data that can be placed in field PIHTDATA. The LLAPI sets this field.

PIHTCURL
A 2-byte fixed field containing the current length of the history data that is in field PIHTDATA. The LLAPI initially sets this field, and your application can update it. If this field is set to the value of zero, the row is deleted.

PIHTDATA
A 32-byte character field containing the history data for this row. The LLAPI initially sets this field, and your application can update it.

PIHTRSV3
A 16-byte area reserved for future use.

PIHT Example

Table 37 on page 136 shows the header of an example PIHT and the following table, Table 38 on page 136, shows the entry field values. The LLAPI creates a PIHT from information you supply in the application program.

Table 37. PIHT Example, Header Field Values

PIHTACRO	PIHTNUMR
PIHT	0009

Table 38. PIHT Example, Entry Field Values

PIHTVER1	PIHTV1JD	PIHTSGRP	PIHTCNTL	PIHTMAXL (DEC)	PIHTCURL (DEC)	PIHTDATA
Y	84123			0016	0010	PERA/SMITH
Y	84127			0016	0008	469-6111
		Y	Y	0016	0013	DATM/05/06/1997
			Y	0016	0010	TIMM/15:20
			Y	0016	0010	USER/JONES
				0016	0009	STAC/OPEN
		Y	Y	0016	0010	TIMM/15:21
				0016	0008	GROA/T53
				0016	0010	CONTROLLER

Program Interface Pattern Table (PIPT)

The program interface pattern table (PIPT) contains the validation criteria used to verify response data in a Tivoli Information Management for z/OS operation. There can be more than one validation pattern for each field. There is one PIPT for each PIDT, and the name of the PIPT is stored in the PIDT field PIDTPTNM. When a PIPT obtain pattern table transaction (T004) runs, the API allocates storage and constructs the PIPT. The PIPT is freed when your application starts a free PIDT transaction (T006) or a free pattern table transaction (T005). The API stores the address of the PIPT in PIDT field PIDTPIPT. Entry type transactions also use the PIPT if you set PIDTGRPX to Y. The table build utility creates the PIPT, or else the PIPT is generated from data attribute and data validation records when a PIDT is generated from a data view record.

For dynamic PIDTs, a PIPT is created dynamically only when processing group prefixes. This PIPT cannot be used for validation. For more information on validation, see Field Validation Using the Field Validation Module BLGPPFVM.

Table 39 shows the structure of the PIPT and the page number where the table fields are explained:

Note: As shown in the table, fields are set by the interface. Your application should *not* attempt to set these fields. If it does, results are unpredictable. In this case, *interface* can mean either the API or the table build utility.

Table 39. Pattern Table (PIPT)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
HEADER			**PATTERN TABLE HEADER 1st RECORD OF TABLE MEMBER**		

Table 39. Pattern Table (PIPT) (continued)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
PIPTACRO	0(0)	4	Acronym of PIPT (character)	Interface	137
PIPTNAME	4(4)	8	Name of this table (character)	Interface	137
PIPTNUMR	12(C)	4	Number of rows in table (fixed)	Interface	137
PIPTVALE	16(10)	1	Indicates if there are validation entries associated with the PIPT.	Interface	137
PIPTRESV	17(11)	63	Reserved	--	137
ENTRY ROW			**PATTERN TABLE ROW 2nd AND SUBSEQUENT ROWS OF TABLE**		
PIPTSYMB	0(0)	5	Field symbolic name (character)	Interface	137
PIPTTYP	5(5)	1	Validation pattern data type - (character) X=expression, A=automatic	Interface	138
PIPTAUTH	6(6)	2	Authorization code index (character)	Interface	138
PIPTPFXL	8(8)	4	Length of p-word field (fixed)	Interface	138
PIPTPATL	12(C)	4	Length of pattern data (fixed)	Interface	138
PIPTPFXI	16(10)	2	Internal form of p-word index (hex)	Interface	138
PIPTPRFX	18(12)	6	P-Word (character)	Interface	138
PIPTDATA	24(18)	32	Validation pattern data (character)	Interface	138
PIPTFLAG	56(38)	1	Pattern process flag field (binary)	Interface	138
PIPTRSV1	57(3B)	23	Reserved	--	138

The following section describes the purpose of each field of a PIPT header:

PIPTACRO
A 4-byte character field containing the string PIPT to identify this table. The table build utility sets this field.

PIPTNAME
An 8-byte character field containing the name of this table. The table build utility sets this field.

PIPTNUMR
A 4-byte fixed field containing the number of rows in this table. The table build utility sets this field.

PIPTVALE
A 1-byte field set by the table build utility. A **Y** in this field indicates that the PIPT will need to be reprocessed with validation records. This means that at least one PIDT entry contains a validation record ID s-word or validation record ID.

PIPTRESV
A 63-byte area reserved for future use.

The following section describes the purpose of each field of a PIPT row:

PIPTSYMB
A 5-byte character field containing the symbolic name of the PIDT response field that uses this pattern. The table build utility sets this field.

PIPTTYP
A 1-byte character field containing a code indicating how to use validation data. Validation data can be a pattern of character data used as a literal expression, or it can be a relation to a Tivoli Information Management for z/OS process, such as a name or user ID response supplied automatically. This field contains an X for expression or an A for automatic. Automatic data is available in interactive Tivoli Information Management for z/OS by the `=attribute` pattern, such as `=DATE`, and `=TIME`. The API provides no means to supply automatic data to your application. Your application must provide automatic data for itself by whatever data source mechanism it can use, such as language built-in functions that provide time and date. The table build utility sets this field.

PIPTAUTH
A 2-byte character field containing the authority code associated with this pattern. The table build utility sets this field.

PIPTPFXL
A 4-byte fixed field containing the length of the p-word stored in field PIPTPRFX. The table build utility sets this field.

PIPTPATL
A 4-byte character field containing the length of the pattern data stored in field PIPTDATA. The table build utility sets this field.

PIPTPFXI
A 2-byte character field containing the internal form (0000 - FFFF) of the p-word index. The table build utility sets this field.

PIPTPRFX
A 6-byte character field containing the p-word. The table build utility sets this field.

PIPTDATA
A 32-byte character field containing the validation pattern data. This is the same pattern data used in Tivoli Information Management for z/OS panels. Refer to the *Tivoli Information Management for z/OS Panel Modification Facility Guide* for information on validation pattern data. The table build utility sets this field.

PIPTFLAG
A 1-byte binary field indicating that this pattern table row begins or ends a p-word group. When this field contains X'40', it indicates that this table row contains a p-word that begins group prefixing. When this field contains X'60', it indicates that this table row contains a p-word that ends group prefixing. The table build utility sets this field.

PIPTRSV1
A 23-byte area reserved for future use.

PIPT Example

Table 40 and Table 41 on page 139 show an example of PIPT rows with entry field values indicated. The table build utility creates a PIPT from information you supply in the utility job input stream.

Table 40. PIPT Example, Header Field Values

ACRO	NAME	NUMR
PIPT	PIDT001P	0009

Table 41. PIPT Example, Entry Field Values

PIPTSYMB	PIPTTYP	PIPTPFXL	PIPTPATL	PIPTPRFX	PIPTDATA
S0B2D	A (auto)	0003	0006	PH/	=PHONE
S0B2D	X (exp)	0003	0005	PH/	IIV12
S0B59	A (auto)	0005	0005	PERS/	=NAME
S0B59	X (exp)	0005	0005	PERS/	CCV14
S0B9B	X (exp)	0005	0005	GROS/	CCV14
S0BE7	X (exp)	0005	0005	PERC/	CCV14
S0BEE	X (exp)	0005	0006	STAC/	<OPEN>
S0C3D	A (auto)	0005	0005	DATO/	=DATE
S0C3D	X (exp)	0005	000C	DATO/	NN</>NN</>NN

Program Interface Argument Table (PIAT)

The program interface argument table (PIAT) contains a list of freeform arguments used in an inquiry. Freeform arguments in the PIAT are specified in the same way as interactive freeform arguments. Each argument is located in an individual row of the PIAT. Your application can append argument data to response p-words to form a prefixed argument. The application can retrieve the p-word used with the argument data from PIDT field PIDTPFXD. The PIAT facilitates range searching and specific argument ordering. Arguments can have leading Boolean or range characters. (If you use Boolean or range characters, they *must* appear as the first character.) The API allocates the PIAT when your application specifies a requested PIAT row count in PICA field PICAREQR and performs an Obtain Inquiry Resource transaction (T106). The PIAT is freed when your application runs a free PIDT transaction (T006). See “Record Inquiry (T107)” on page 84 for more information about a PIAT.

The length of the field PIATDATA is 33 characters. The maximum number of characters available from a freeform argument segment for use in an inquiry is limited to the length of the key used to define the SDIDS. For example, if your application is searching a database with an SDIDS defined with a 32-byte key, a maximum of 32 bytes of each freeform argument segment is used to perform the inquiry.

Table 42 shows the structure of the PIAT and the page number where the table fields are explained:

Note: As shown in the table, some fields are set by the interface. Your application should *not* attempt to set these fields. If it does, results are unpredictable.

Table 42. Argument Table (PIAT)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
HEADER			**ARGUMENT TABLE HEAD 1st RECORD OF TABLE**		
PIATACRO	0(0)	4	Acronym of PIAT (character)	Interface	140
PIATNUMR	4(4)	4	Number of table rows (fixed)	Interface	140
PIATNARG	8(8)	4	Number of arguments (fixed)	Application	140

Table 42. Argument Table (PIAT) (continued)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
PIATRESV	12(C)	12	Reserved	--	140
ENTRY ROW			**ARGUMENT TABLE ROW 2nd AND SUBSEQUENT ROWS OF TABLE**		
PIATDATL	0(0)	4	Length of argument data (fixed)	Application	140
PIATDATA	4(4)	33	Argument data (character)	Application	140
PIATRSV1	37(25)	3	Reserved	--	140

The following list describes the purpose of each field of a PIAT header.

PIATACRO
A 4-character field containing the character string PIAT to identify this program interface argument table. The LLAPI sets this field.

PIATNUMR
A 4-byte fixed field containing the number of rows in this table. The LLAPI sets this field.

PIATNARG
A 4-byte fixed field containing the number of argument rows to process in this table. Your application sets this field.

PIATRESV
A 12-byte area reserved for future use.

The following list describes the purpose of each field of a PIAT row.

PIATDATL
A 4-byte fixed field indicating argument data length in this PIAT row. If this field is 0, the API does not perform argument collection. Your application sets this field.

PIATDATA
A 33-byte character field containing the freeform argument data in this PIAT row. Your application enters data in the same way you would enter argument data interactively with the p-words preceding the argument data. If you do not enter a p-word, the API uses the argument as entered. Searching on abstract data is an example. The argument must be left-justified with no imbedded blanks. When you are using Boolean or range operators, they must appear in the first position. The application sets this field.

PIATRSV1
A 3-byte area reserved for future use.

PIAT Example

Table 43 and Table 44 on page 141 show an example of a PIAT with 9 rows, 2 of which are used.

Table 43. PIAT Example, Header Field Values

ACRO	NUMR	NARG
PIAT	9	2

Table 44. PIAT Example, Entry Row Values

PIATDATL	PIATDATA
000A	TIMA/13:00
000B	-TIMA/14:00
000B	¬TIMA/13:30
000B	¦TIME/14:15
000C	TRMID_VID009
0008	TERMINAL
0006	SMOKES
0004	WHEN
0003	HOT

Program Interface Results Table (PIRT)

The program interface results table (PIRT) contains a list of external record IDs found that meet specific search criteria. For each record processed, the API returns the record type, data from one field of the record (if requested), and an error code. The API allocates and builds a PIRT for each inquiry transaction it performs. The API stores the PIRT address in PICA field PICAPIRT. Your application frees a PIRT by performing a free PIRT transaction (T007). The LLAPI frees it automatically if the PIRT allocated for a prior inquiry is not large enough to satisfy the results of the current inquiry.

Table 45 shows the structure of the PIRT and the page number where the table fields are explained:

Note: As shown in the table, fields are set by the interface. Your application should *not* attempt to set these fields. If it does, results are unpredictable.

Table 45. Results Table (PIRT)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
HEADER			**RESULTS TABLE HEADER 1st RECORD OF TABLE**		
PIRTACRO	0(0)	4	Acronym of PIRT (character)	Interface	142
PIRTROWS	4(4)	4	Number of results table rows allocated (fixed)	Interface	142
PIRTHITC	8(8)	4	Number of results table entries (fixed)	Interface	142
PIRTSRRC	12(C)	4	Number of results from search (fixed)	Interface	142
PIRTBHIT	16(10)	4	The match index of the first match found (fixed)	Interface	142
PIRTRESV	20(14)	60	Reserved	--	142
ENTRY ROW			**RESULTS TABLE ROW 2nd AND SUBSEQUENT ROWS OF TABLE**		
PIRTRNID	0(0)	8	Record identifier (character)	Interface	142
PIRTINDX	8(8)	4	Record type index (character)	Interface	142
PIRTDATL	12(C)	4	Length of associated data (fixed)	Interface	142

Table 45. Results Table (PIRT) (continued)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
PIRTDATA	16(10)	45	Record associated data (character)	Interface	142
PIRTCODE	61(3D)	2	Record processing code (character)	Interface	143
PIRTRSV1	63(3F)	17	Reserved	--	143

The following list describes the purpose of each field of a PIRT header.

PIRTACRO
A 4-character field containing the character string PIRT to identify this program interface results table. The LLAPI sets this field.

PIRTROWS
A 4-byte fixed field containing the number of PIRT table rows allocated. If a value is specified on the session parameters for SORTPFX-N1, then the number of rows allocated is limited to that value. The LLAPI sets this field.

PIRTHITC
A 4-byte fixed field containing the number of PIRT table rows used when an inquiry produces matches. If you specify a value on the session parameters for SORTPFX-N1, then the number of matches returned in the PIRT is limited to that value. In that case, the value of PIRTHITC can be less than or equal to the value in SORTPFX-N1. The LLAPI sets this field.

PIRTSRRC
A 4-byte fixed field containing the number of matches when an inquiry transaction runs. This value can be larger than PIRTHITC if you specify a value for SORTPFX-N1 in the session parameters. The LLAPI sets this field.

PIRTRESV
A 64-byte area reserved for future use.

The following list describes the purpose of each field of a PIRT row.

PIRTBHIT
A 4-byte fixed field indicating the beginning match number returned. If your application specifies zero in PICABHIT, the API uses a value of one.

PIRTRNID
An 8-byte character field containing the external record ID of a found record that matches the search criteria. The LLAPI sets this field.

PIRTINDX
A 4-byte fixed field containing the record type index (s-word index) that identifies the record type without the leading S. The LLAPI sets this field.

PIRTDATL
A 4-byte fixed field containing the length of the data in the associated data field (PIRTDATA). The LLAPI sets this field.

PIRTDATA
A 45-byte character associated data field that contains up to 45 bytes of data extracted from a unique field in the record. You identify which field is to be extracted from the record by putting that field's s-word index into PICA field

PICASRCH before your application performs the inquiry transaction. You can choose a different associated data field for each inquiry. Nothing is returned in the associated data field if you specify a text or list data item. The LLAPI sets this field.

PIRTCODE
A 2-byte character field containing a code that indicates record processing results for a particular match. The LLAPI returns one of the following codes:

00 No error detected

01 The record found a read error

02 The record was not found

03 The record being updated was not available

04 The record was busy

05 Not enough storage to read in record

06 Unknown problem when reading record.

PIRTRSV1
A 17-byte area reserved for future use.

PIRT Example

Table 46 shows an example of PIRT with entry rows that depict two problem records found by a search. It is assumed that PICASRCH is set to S0E0F before your application requests the inquiry transaction. Setting this value causes the description abstract to be returned with each record found by the search. PIRT HEADER fields are not shown.

Table 46. PIRT Example, Entry Field Values

PIRTRNID	PIRTINDX	PIRTDATL	PIRTDATA	PIRTCODE
RECID001	0032	X(0018)	TERMINAL SMOKES WHEN HOT	00
RECID002	0032	X(0010)	DISPLAY UNIT BAD	00

Program Interface Message Block (PIMB)

If you set PICAMSGD to C or B at the time of initialization, then messages are returned in the Program Interface Message Blocks (PIMBs). A PIMB defines the format of a message block on the message chain. The PICA field PICAMSGP points to the first PIMB on the message chain. The API is responsible for allocating and freeing message chain blocks.

Note: Only Tivoli Information Management for z/OS messages are chained. API messages are not chained but are written to the API Print data set when PICAMSGD is set to P or B.

Table 47 shows the structure of the PIMB and the page number where the table fields are explained:

Note: As shown in the table, fields are set by the interface. Your application should *not* attempt to set these fields. If it does, results are unpredictable.

Table 47. Message Block (PIMB)

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
PIMBFWDP	0(0)	4	Pointer to next block (pointer)	Interface	144
PIMBPRVP	4(4)	4	Pointer to previous block (pointer)	Interface	144
PIMBDATL	8(8)	4	Length of message data (fixed)	Interface	144
PIMBDATA	12(C)	Variable length	Message data (character)	Interface	144

The following section describes the purpose of each field of a PIMB:

PIMBFWDP
A 4-byte pointer to the next message block. The LLAPI sets this field.

PIMBPRVP
A 4-byte pointer to the previous message block. The LLAPI sets this field.

PIMBDATL
A 4-byte fixed field containing the length of the message data in this block. The LLAPI sets this field.

PIMBDATA
A variable length character field containing the message data in this block. The LLAPI sets this field.

PIMB Example

Table 48 shows an example of a PIMB chain.

Table 48. PIMB Example

PIMBFWDP	PIMBPRVP	PIMBDATL	PIMBDATA
Next	0000	Length of message	Text of message
0000	Previous	Length of message	Text of message

3

Using the HLAPI

This chapter tells you how your applications can use the HLAPI to access a Tivoli Information Management for z/OS database to perform these tasks:

- Creating records
- Updating records
- Retrieving records
- Inquiring about IDs (and associated data) of records meeting specified data search criteria
- Deleting records
- Using TSPs

This chapter discusses only the interface that the HLAPI provides to your application. The methods that your application uses to collect data and process transaction results are not discussed.

The HLAPI/REXX interface enables you to set up and use an HLAPI session from a REXX program; however, you must understand the types of data that are input to and returned from the HLAPI. The HLAPI/REXX interface is so closely tied to the HLAPI that it is also explained in this chapter.

The HLAPI performs its work by using the LLAPI. One HLAPI transaction can result in multiple LLAPI transactions being performed. You might have to tailor the LLAPI TSPs. See "LLAPI Operating Characteristics" on page 15 for more information.

HLAPI Operating Characteristics

The is tied to the LLAPI because it uses LLAPI transactions, so the same LLAPI limitations and characteristics apply. The API environment requires that MVS/ESA, data management services, and VSAM be available.

If you are using the HLAPI from remote platforms, some of the operating characteristics are different. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for more information about remote HLAPI characteristics.

Control Transfer Considerations

In an MVS environment, most high-level languages create an internal parameter list structure in which the first 4 bytes are the address of the HICA. The call to the server passes the HICA structure itself and not its address. See Figure 6 on page 149 for more information.

Consider how to transfer control to the HLAPI server (module BLGYHLPI). BLGYHLPI is installed with attributes of AMODE 31, RMODE ANY. Also consider establishing the server entry address by preloading BLGYHLPI using the MVS

LOAD macro or equivalent service in the language you are using. This method is usually more efficient because the server loads into storage only once, which saves load I/O cycles. Consider, too, whether you want to enable the HLAPI to return data above the 16MB address range.

Operating Mode

Transactions are synchronous in that your application cannot request or start another HLAPI transaction until the previous one completes. "Initialize Tivoli Information Management for z/OS (HL01)" on page 153 contains additional information on transaction details.

Validating Data

The HLAPI does not automatically perform response validation as do panel dialogs in Tivoli Information Management for z/OS entry or inquiry mode. The HLAPI can optionally use the LLAPI's field validation module (BLGPPFVM) to validate field response data on a field-by-field basis. The HLAPI can also perform some validation processing with the = sign. Data from validation and attribute records can be used to construct PIPTs and thus be used for validation. You can also define another field in the record that names a validation record to use for validating field data. With regard to equal sign processing for the HLAPI, if data is specified with an equal sign, then the API will attempt to process it. A PDB EQUAL_SIGN_PROCESSING should be set to YES in the HLAPI to specify equal sign processing. The four patterns currently supported in the API environment are:

- DATE
- TIME
- USER
- CLASS

Collecting Data in Mixed Case

Data which is not validated is passed through the API in the case in which your application supplies it. To convert the data to the case specified in the PIDT (derived from the assisted-entry panel or data attribute record for the field), you specify data validation for each field you want to convert. Data can be converted as part of data validation processing performed by the HLAPI using module BLGPPFVM.

Loading and Initializing

Your application must establish program linkage to the server routine BLGYHLPI before you can initiate the Tivoli Information Management for z/OS environment. You initiate the Tivoli Information Management for z/OS environment by using the initialize Tivoli Information Management for z/OS transaction (HL01) to call the API. The HLAPI runs all other transactions only after your application initializes the Tivoli Information Management for z/OS environment.

Structure and Processing

The HLAPI uses a common data structure to pass data parameters (not program parameters) between your application and the server. This data structure, called a Parameter Data Block (PDB), is described in detail later in "HLAPI Structures" on page 216. The HLAPI, like the LLAPI, is also defined as a data interface that enables Tivoli Information Management for z/OS data access functions. The interface supports structured and text data processing through the HLAPI implementation of the LLAPI. A set of PDBs passes data for all interface functions.

Terminating

To end the Tivoli Information Management for z/OS environment, your application calls the API using the environment termination transaction (HL02). This transaction

frees up any resources held by Tivoli Information Management for z/OS. Your application must then delete the server routine if your application loaded the server routine earlier.

Addressing

Applications using the API can reside in an address space above or below the 16MB address range. The MVS address space environment can be TSO, non-TSO, or MVS/ESA batch. The components of the interface all reside above the 16MB address range. Applications that use either 24-bit or 31-bit addressing can call the server.

If your application runs below the 16MB address range, it must use the MVS LINK macro when it transfers control to the server to maintain correct address mode.

The HLAPI allocates storage and returns data using addresses above the 16MB address range if you specify the HIGH_MEMORY PDB when you initialize the HLAPI.

Checking Records In and Out

Checking out a record with an API differs from what interactive users of Tivoli Information Management for z/OS are used to. When you check out a record with the API, it remains checked out and unavailable to anyone else, until you perform a check in transaction, until an optional administrator-specified time limit is reached, or until an administrator manually checks in the record. This way, you can be sure that the record you want to work with is unchanged from the time you find it until the time you make your own changes to it, even if your application ends before checking in the record. Your system administrator can define an expiration time interval that will, in effect, check in records after the specified period of time. See "Check Out Record (HL04)" on page 162 for additional information about checking out records.

If your application runs a check out transaction for any record, be sure to check it back in when you finish with it.

Note: If you fail to check in a record, the system administrator can check it in interactively. Refer to the *Tivoli Information Management for z/OS User's Guide* for details on database cleanup.

HLAPI Environment Considerations

Your application must call the server BLGYHLPI in problem program state with storage key 8 under control of a task that was attached with storage key 8. If it does not, unpredictable results occur.

NetView Considerations

If your application runs under NetView, all Tivoli Information Management for z/OS components must be put in an authorized program facility (APF) library. Each service request is a transaction.

HLAPI Inquiries

Database inquiry uses a single argument collection mechanism rather than two as used in the LLAPI. The HLAPI determines which HLAPI inquiry process to use by the way the data is specified to the HLAPI. The HLAPI extracts the resulting list of record IDs and associated data from the program interface results table (PIRT) and uses PDBs to pass the list back to your application.

Using Alias Names and Default Data

The HLAPI enables you to define alias names for each data field, thus eliminating

the need to use s-word or p-word indexes. You can also define alias names for Tivoli Information Management for z/OS prefixes used in database inquiries. Alias names are defined in tables that are stored in the same data set that contains LLAPI tables (static PIDTs). Using an alias name for a data field is optional. Your application can specify different alias tables depending on the needs of a transaction.

The HLAPI also enables you to define alias names for Program Interface Data Table (PIDT) member names and data view names.

Note: User defined alias names are considered keywords and cannot handle double-byte character set (DBCS) data.

When creating records, the HLAPI lets you store default field responses in the database. Default field responses are stored in the alias table. The HLAPI incorporates LLAPI alias table processing with various default processing options.

Data Model Considerations

You must define a "data view" for your application for those transactions that require or return Tivoli Information Management for z/OS data. You can use static PIDTs built by the Table Build Utility (BLGUT8) or data model records to define this view. See "Field Validation Using the Field Validation Module BLGPPFVM" on page 279 for additional information on data model records.

History Data Considerations

History data processing is provided in two independent parts. The first allows history data to be returned on the output PDB chain if the optional HISTORY_DATA control PDB is specified as part of the retrieve record transaction (HL06). No additional level of authority is required for this function beyond that for record retrieval.

The second part of history processing provides the ability to delete history entries. It involves both the RETRIEVE (HL06) transaction and the UPDATE (HL09) transaction. In order to perform the delete function the API user must have database administrator authority. Details about how to utilize the history features using the HLAPI is contained in "Retrieve Record (HL06)" on page 171 and "Update Record (HL09)" on page 183.

Record File Processing

The HLAPI uses the LLAPI to perform transactions. You can use two modes of operation: panel processing and bypass panel processing. If you use panel processing, the HLAPI (via the LLAPI) performs record file processing for create and update transactions by using selection 9 File Record on summary panels. If your application uses a selection other than 9 to file a record, see "Tailoring the Application Program Interfaces" on page 289 for information on customizing your application. Record files are processed just as if you used the panel interface. Certain data fields, such as Date last altered, Time last altered, and Time entered, are automatically collected by Tivoli Information Management for z/OS. If you use bypass panel processing, the HLAPI (via the LLAPI) uses user exits to file records.

Logical Database Partitioning

If you are using logical database partitioning, you can perform the database access transactions (retrieve, update, check in, check out, add record relation, and delete) only for records whose Owning Partition matches the Primary Partition of your privilege class. API applications cannot perform multipartition searches.

Date Considerations

Dates used by your application can be processed in either of two ways:

Database format

Dates are passed to your application from the API in the default external date format. Dates your application passes to the API must be in either the default format or, if one is defined, the old format specified in the session parameters being used. Dates passed in either format are automatically converted to internal format when they are stored in the SDDS portion of the database.

Application-specified format

Dates are passed between the API and your application in a date format your application specifies. This format does not need to match that of the database. The API automatically converts dates from the internal format in the database to the format you specify when passing data to your application and from your specified format to the database's internal format when receiving data from your application.

An application-specified date format is set in the HLAPI by specifying the desired date format (for example, MM/DD/YYYY or YYYY.MM.DD in a control PDB with a PDB name of DATE_FORMAT.

Database date format is the default and can be specified in the HLAPI by specifying a control PDB DATE_FORMAT with a value of DATABASE or by never specifying a control PDB named DATE_FORMAT.

Understanding LLAPI Operating Characteristics

Because the HLAPI uses the LLAPI, you might need to understand some LLAPI operating characteristics. See the following for more information:

- "API Control Flow" on page 283
- "Exit and Terminal Simulator Limitations" on page 17
- "Record Update Retry and Wait Considerations" on page 18
- "NetView Considerations" on page 19
- "LLAPI Logic" on page 17

HLAPI Calls

This example shows the HLAPI interface call syntax which uses call-with-parameter-list notation.

```
<Label> CALL BLGYHLPI
```

Figure 6 shows the parameter list structure used for calling the API as it appears to an assembler language program. The parameter list points to the HLAPI communications area (HICA).

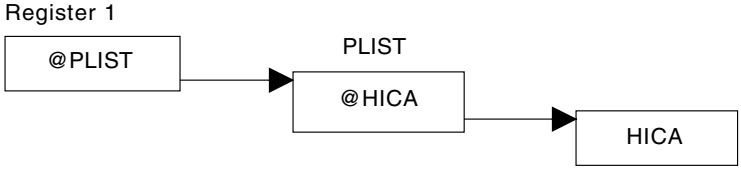

Figure 6. Input Parameter List for the HLAPI

Data Sets

The HLAPI uses the following data sets:

- **Text data set**

  This data set stores text data for a unique text type. (See "Data Sets" on page 21 for a description of a text data set.) This data set is required if your application uses the data set format of adding or reading text; it is not required if your application uses the buffer method to add or read text.

- **HLAPILOG data set**

  This data set contains transaction activity messages created by the HLAPI. The data set is a sequential, non-VSAM data set that you write to a system output device, tape, or direct access volume.

  DCB parameters for this data set are:
  DSORG = PS
  RECFM = VBA
  LRECL = 125
  BLKSIZE = 6144

  If you do not preallocate HLAPILOG and you request logging, the HLAPI dynamically allocates HLAPILOG to a SYSOUT=A data set.

  Note: The HLAPI stops logging transaction activity messages if an error occurs while writing to this data set.

- **Report format table data set**

  A description of this data set can be found on page 23.

- **SYSPRINT data set**

  A description of this data set can be found on page 23.

- **APIPRINT data set** This data set is used if the control PDB named APIMSG_OPTION is set to B or P on the initialization transaction.

  A description of this data set can be found on page 24.

- **SYSUDUMP data set**

  A description of this data set can be found on page 24.

Errors and Messages

The HLAPI returns messages to your application from the API subtask and the LLAPI through the HICA. HICARETC contains return codes and HICAREAS contains reason codes. If your application requests message chaining, HICAMSGP points to the first PDB on the message chain. HICAERRP points to the first PDB on the error chain containing codes from PIDTCODE in the PIDTor codes from validation routines. A list of validation codes can be found on page 236. The interface writes HLAPI messages to the HLAPILOG data set, the message PDB chain, or both.

Structures

The structures used by the HLAPI are described later in this chapter. For information about specific structures, see:

- "High-Level Application Program Interface Communications Area" on page 216 for the HICA
- "Parameter Data Block" on page 218 for the PDBs

- "Alias Tables" on page 238 for the alias table

HLAPI Transactions

These are the three types of HLAPI transactions that your application uses:

- Environment control to establish and to end the Tivoli Information Management for z/OS environment and the HLAPI.
- Service to obtain services such as checking in and checking out records being modified by the HLAPI and deleting special data sets.
- Database access to perform the database tasks listed at the beginning of this chapter.

Table 49 lists each function that the HLAPI performs, its associated transaction number, and the page in this book where you can find more information about the function.

Table 49. HLAPI Functions and Transaction Numbers

HLAPI Function	Transaction Number	Transaction Type	page
Initialize Tivoli Information Management for z/OS	HL01	Environment Control	153
Terminate Tivoli Information Management for z/OS	HL02	Environment Control	160
Obtain external record ID	HL03	Interface Service	161
Check out record	HL04	Interface Service	162
Check in record	HL05	Interface Service	164
Retrieve record	HL06	Database Access	171
Reserved	HL07	—	—
Create record	HL08	Database Access	178
Update record	HL09	Database Access	183
Change record approval	HL10	Database Access	191
Record inquiry	HL11	Database Access	194
Add record relation	HL12	Database Access	202
Delete record	HL13	Database Access	205
Start user TSP or TSX	HL14	Environment Control	166
Free text data set	HL15	Environment Control	169
Delete text data set	HL16	Environment Control	170
Get Data Model	HL31	Interface Service	207

The remainder of this chapter, starting at "Environment Control Transactions" on page 153, describes the use of these transactions. For each transaction, introductory text describes required and optional structure fields, such as control blocks and PDBs, their value settings, and their relationships to other structures. A table shows transaction flow from the application through the HLAPI and back to the application. The flow tables in this chapter describe the actions of your application and the server for each of the HLAPI transactions. This information describes pertinent relationships between HLAPI structure fields and data PDBs that can aid you in understanding the HLAPI in terms of what your application must do. At the end of the chapter are examples of the setup for some of the HLAPI transactions.

You use the HLAPI by setting fields in the HLAPI communications area structure (HICA), building chains of control PDBs for transaction control and input PDBs for data input, and calling the server module BLGYHLPI. Your call passes the HICA control block to the server as a parameter.

See "HLAPI Structures" on page 216 for more information on HLAPI structures and PDBs.

Control PDB

You must create and specify a control PDB chain (linklist) for all transactions. The PDBs on this chain contain data that identifies the requested transaction and tells the HLAPI how to process it. Your application can manage PDB chains in one of two ways:

- For a given transaction, your application can build a new control PDB chain, use it, free or deallocate it, then start over again for the next transaction by building a new control PDB chain
- A method that eliminates constant reallocation of PDBs is to build a control PDB chain that contains PDBs for all control parameters. For each transaction, set the PDBDATL field to zero for any PDBs you do not want processed, and ensure that the PDBDATL field is set to the length of the data for the PDBs that you do want processed. This method takes advantage of the fact that the HLAPI ignores a control PDB whose PDBDATL field is set to zero when transaction processing begins.

For each transaction, the HLAPI looks for specific control PDBs and ignores other PDBs even if they exist on the control PDB chain. See the individual transaction descriptions for information about which control PDBs the HLAPI uses for each transaction.

Input PDB

Some transactions (for example, update, create, and inquiry) require you to build an input PDB chain for HLAPI processing. The input PDB chain contains the data to be processed in the form of search criteria for inquiry transactions, and data to be added to a Tivoli Information Management for z/OS record for update and create transactions. If you request validation for input data, the HLAPI validates all PDBs marked to be validated in the input PDB chain at one time and sets the input PDB field PDBCODE to a nonblank character if that PDB has an error. The return code and reason code fields returned for the transaction apply only to the first PDB with an error. The HLAPI returns error information in all input PDBs that contain errors. If one input PDB contains several errors, you must correct the first error and then run your program again to find any further errors in that input PDB.

Output PDB

The API returns transaction processing results to the output PDB chain. Each PDB in the chain contains one item of data. For example, if three record identifiers are returned as the result of a search of the database, the HLAPI allocates three output PDBs and stores these record identifiers in them. Your application must process information in the output PDB chain before starting another transaction because the output chain is always freed at the next transaction startup.

Message and Error PDB

The processing of each HLAPI transaction results in the setting of fields HICARETC and HICAREAS. If these fields are not 0, some type of error has occurred. For successful transactions, the HLAPI can return information to a message PDB chain pointed to by HICAMSGP. For unsuccessful transactions, the HLAPI can return information to both a message and an error PDB chain pointed to by HICAMSGP and HICAERRP. You must

process this information before your application starts the next transaction because the HLAPI frees these chains at transaction startup time.

Environment Control Transactions

Use this group of transactions to initialize and end the Tivoli Information Management for z/OS environment. You can also establish particular operating characteristics for the environment. The environment control transactions are HL01 and HL02.

Initialize Tivoli Information Management for z/OS (HL01)

This transaction initializes the Tivoli Information Management for z/OS environment. It starts the HLAPI and the LLAPI, and it prepares Tivoli Information Management for z/OS for further transaction processing. HICA field HICAENVP identifies the HLAPI environment to your application, and your application never sets this field. Your application can initialize any number of environments, but each environment must use a unique HICA and a unique application ID. Therefore, your application can use many different HLAPI sessions, each with its own processing options defined at initialization time. If you have multiple sessions and are using logging for each one, allocate the log to SYSOUT so you do not lose information from any of the sessions. You can specify a different identifier to be included in certain log messages for each session.

Note: In an application environment having multiple active HLAPI sessions, the API serializes transactions started from different HLAPI sessions, so they run in FIFO (first in, first out) order. You end an HLAPI session by starting an HL02 transaction using the HICA associated with the session you are terminating.

If you are using a client connecting to Tivoli Information Management for z/OS through the MRES, you may find initialization time to be significantly improved by using pre-started MRES sessions. The procedure for doing this is described in the *Tivoli Information Management for z/OS Client Installation and User's Guide* in either the chapter "Configuring and Running an MRES with APPC" or the chapter "Configuring and Running an MRES with TCP/IP". If you do use pre-started MRES sessions, logging is controlled by the MRES, and the status of any PDBs specified is as follows:

- If you use pre-started MRES sessions and specify these values in the HL01 transaction, the values that you specify are saved and supplied to subsequent TSXs:
  - APPLICATION_ID
  - DATE_FORMAT
  - PRIVILEGE_CLASS
- If you use pre-started MRES sessions and specify these values in the HL01 transaction, they are ignored:
  - APIMSG_OPTION
  - CLASS_COUNT
  - DEFAULT_OPTION
  - DEFAULT_DATA_STORAGE_SIZE
  - HIGH_MEMORY
  - HLAPILOG_ID
  - HLIMSG_OPTION

  - MULTIPLE_RESPONSE_FORMAT
  - SPOOL_INTERVAL
  - TABLE_COUNT
  - TIMEOUT_INTERVAL
- If you use pre-started MRES sessions and specify these values in the HL01 transaction, they must match the values that you specify for the pre-started session:
  - BYPASS_PANEL_PROCESSING
  - DATABASE_ID
  - SESSION_MEMBER
  - PDB_TRACE

At initialization time, you specify session operating characteristics such as:
- The ID of your application used for this session (you can change the ID on subsequent transactions throughout the life of the session)
- The Tivoli Information Management for z/OS session member name
- The maximum number of alias tables and privilege class records in storage during the session
- Information about how the LLAPI and the HLAPI process messages
- The ID of the Tivoli Information Management for z/OS database used for the session
- The amount of storage allocated to hold default data
- The session identifier for this session to be used on some HLAPI log messages
- Which processing mode to use: regular panel processing or bypass panel processing
- The date format to be used

The API automatically sets the HICAENVP pointer during initialization. Your application must not alter this pointer value for the duration of the session.

Follow these outline steps to initialize Tivoli Information Management for z/OS

1. Define a storage area for the HICA you use in your application.
2. Initialize HICAACRO to *HICA*.
3. Initialize HICAENVP to zero.
4. Initialize HICALENG to the length of the HICA.
5. Allocate and initialize control PDBs with values that govern how the HLAPI environment operates. The HLAPI only processes control PDBs having a nonzero data length. This enables your application to define a chain of control PDBs for use by all transactions. Your application can indicate which PDBs to process by setting the data length field (PDBDATL) of the PDB to a nonzero value.
6. Ensure that HICACTLP points to the first control PDB.
7. Initialize remaining HICA fields to zero.
8. Consider allocating a chain of input PDBs at this time. A working set of PDBs using an adequately sized data area might satisfy all your transaction processing needs for the session. Doing this would eliminate allocation and deallocation overhead for input PDBs.
9. Start the server module BLGYHLPI passing the HICA as a parameter.

The following HICA fields and PDBs are required for this transaction:

HICAENVP

Initialize this 4-byte pointer field to zeros. The server uses this field to store the address of the HLAPI environment used for the duration of the session. Your application must not change the contents of this pointer until after the session ends.

HICACTLP

Each PDB in the chain must have its name in PDBNAME and its data value in PDBDATA. Control PDBs need not be specified in any order. See "Parameter Data Definition" on page 225 for more information about the control PDBs. The following PDBs are required:

- TRANSACTION_ID must contain the 4-character uppercase transaction code of HL01. If you want PDB data tracing for debugging purposes or PDB data logging, set PDBPROC to a value of T. This causes the logging of up to 32 bytes of PDBDATA information for each PDB used throughout the session.

  If you have requested PDB tracing, the contents of all PDBs are stored in the HLAPILOG data set. This method is useful for debugging purposes because the contents of the control, input, output, message, and error PDBs are included. The data portion of the PDB might be truncated.

- APPLICATION_ID must contain a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses for this session. The application ID remains in effect until changed on a subsequent transaction. If APISECURITY=ON is specified in your BLX-SP startup parameters, you must ensure that the MVS user IDs running your application are allowed to use this application ID. "API Security" on page 287 contains additional information regarding API security.

- SESSION_MEMBER must contain a 7- or 8-character uppercase load library session-parameters member name that Tivoli Information Management for z/OS uses for this session.

- PRIVILEGE_CLASS must contain a 1- to 8-byte uppercase startup privilege class name that the API passes to Tivoli Information Management for z/OS. This privilege class remains in effect until changed on a subsequent transaction. A privilege class name can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana. The application ID used on the HL01 must be an eligible user of this privilege class.

The following PDBs are optional:

- TABLE_COUNT must contain a 4-byte fixed value indicating the maximum number of alias tables and PIDT tables that can be in storage during this Tivoli Information Management for z/OS session. Static PIDTs and PIDTs generated from data view records are treated the same for caching purposes (that is, all types are cached).

  When data model records or PIDTs are updated and you want to force the cache to get the new updates, use the BRDCST operator command with the TABLES keyword to pick up the updates. More information on using the BRDCST command can be found in *Tivoli Information Management for z/OS Operation and Maintenance Reference*.

  It can take a significant amount of time to generate a PIDT from data view records. The length of time depends on the number of data attribute records (and validation records that they reference) contained in the data view record. Therefore, it can be especially important to direct the HLAPI to maintain PIDTs in storage if you are

using data model records. PIPTs associated with PIDTs kept in storage remain in storage by default because each PIPT has a PIDT pointing to it.

If you omit this parameter, or if it contains a null or zero value, the HLAPI:

- Performs no alias table processing within this Tivoli Information Management for z/OS session
- Performs no PIDT caching
- Ignores any ALIAS_TABLE control PDB specifications

You can specify any value from 0 to 256.

- CLASS_COUNT must contain a 4-byte fixed value indicating the number of Tivoli Information Management for z/OS privilege class records that can be in storage during this Tivoli Information Management for z/OS session. If you omit this parameter or enter zero as its value, the Tivoli Information Management for z/OS session operates with a single class record in storage at a time.

  Note: If you plan to use more than one privilege class in the session and switch between classes, you can minimize (or eliminate) the I/O involved in bringing the classes in and out of storage by including this PDB and setting its PDBDATA field to the number of classes needed for the session.

- APIMSG_OPTION must contain the single character value of P, C, or B, which specifies the destination of LLAPI messages for the session.
  - P specifies that the API writes LLAPI messages to the APIPRINT data set or to SYSOUT. If APIPRINT is allocated to a data set and that data set becomes full, the LLAPI closes, reallocates, and reopens the APIPRINT data set, and old log messages are lost. If you write LLAPI messages to SYSOUT, you do not lose old log messages.

    Note: You must specify the PDB named SPOOL_INTERVAL to log LLAPI messages, and you can use this PDB to tell the API to close and reopen the data set at a specified interval.
  - C specifies that LLAPI messages pass to the HLAPI.
  - B specifies that the API performs both P and C.

  If you specify any character value other than P, C, or B, or if you omit this parameter, then the API performs option C.

- HLIMSG_OPTION must contain the single character value of P, C, or B defining how the HLAPI processes LLAPI messages for this session.
  - P specifies that the HLAPI writes messages to the HLAPILOG data set. If the data set becomes full, the HLAPI closes, reallocates, and reopens the data set, and old log messages are lost.

    Note: You must specify the PDB named SPOOL_INTERVAL to log LLAPI messages, and you can use this PDB to tell the HLAPI to close and reopen the data set at a specified interval.
  - C specifies that the HLAPI chains messages on the message PDB chain.
  - B specifies that the HLAPI performs both P and C.

  If you specify any character other than P, C, or B, or if you omit this parameter, then the HLAPI performs option C.

- TIMEOUT_INTERVAL must contain a 4-byte fixed field value. This value specifies the interval of time in seconds that a transaction can run before a timer interrupt occurs. If a timeout occurs, the HLAPI terminates its LLAPI session and terminates the HLAPI session. You must perform another initialize Tivoli Information Management for z/OS transaction (HL01) before you can perform additional transactions. If you omit this parameter, the HLAPI defaults to a value of 300. If you specify an interval of greater than 0, but less than 45 seconds, the interval is set to 45 seconds.
- SPOOL_INTERVAL must contain a 4-byte fixed field value. This value specifies the interval of time in minutes between the instances when the API spools activity logs (APIPRINT and HLAPILOG data sets) when it prints messages. After the interval expires, the API closes and reopens the data sets, and printing starts again at the beginning of the data sets.

  The maximum number of minutes you can use is 60*24 (that is, 60 minutes multiplied by 24 hours=1440 minutes, one full day). If you specify more minutes than there are in a day, the activity log closes and reopens after 1440 minutes, ignoring your specification.

  If you omit this parameter, the APIs do not perform logging regardless of the specifications in the HLIMSG_OPTION or APIMSG_OPTION PDBs.
- DATABASE_ID must contain the name or ID number of the database your application uses for the session. Subsequent retrieve or entry transactions access the database you identify in this PDB. The database ID for Tivoli Information Management for z/OS is 5. If you do not specify a value for DATABASE_ID, the HLAPI automatically sets the database ID to 5.
- DEFAULT_OPTION must contain character data that defines how the HLAPI performs default data response processing when records are created in this session. Using the default data processing option enables you to obtain predefined default data from an alias table for PIDT data fields that your application does not provide a response for. The valid data values for DEFAULT_OPTION are REQUIRED, ALL, and NONE.
  - REQUIRED specifies that only required fields are candidates for default response processing. Required fields, if any, are defined in the PIDT used for a specific transaction.
  - ALL specifies that all response fields are candidates for default response processing.
  - NONE indicates that the HLAPI performs no default response processing.

  The HLAPI can combine data obtained from input PDBs with default data to create the record. Input PDB data always overrides default data. If you omit this parameter, the HLAPI assumes a value of NONE.
- DEFAULT_DATA_STORAGE_SIZE must contain a 4-byte fixed field that specifies how much additional storage the API allocates to hold default response data when using an alias table data when creating records. If you plan to use default data when creating records and the total size of the data is greater than 1024 bytes, you must include this PDB and set PDBDATA to a value (in bytes) larger than the total size of the data. When the HLAPI creates records, it calculates the size of the response buffer it requires by totaling the lengths of all the input data PDBs and adding the specified default data storage size or, if that is not specified, a default value of 1024

bytes. This calculation allows room to store default data in the response buffer. When the HLAPI performs create response processing, it always checks to make sure the response does not overlay storage. If the HLAPI check indicates that the response would overlay storage, the HLAPI transaction terminates with an error code.

- HLAPILOG_ID must contain a 1- to 8-character HLAPI session identifier that you can specify to identify the session in HLAPI log file messages. If you do not specify a value for HLAPILOG_ID, then this field is blank in HLAPI log file messages. Below are some examples of messages with a HLAPILOG_ID specified as *sessid* as they appear in the HLAPI log file:

  ```
  DATE: MM/DD/YY  TIME: HH:MM:SS HIGH-LEVEL INTERFACE ACTIVITY LOG FOR APPL: applid
  SESSION ID: sessid PAGE: nnnnn

  BLG25013I sessid THE HLAPI WAS STARTED FOR APPLICATION applid ON date AT time.

  BLG25015I sessid TRANSACTION trans PROCESSED.  RETURN CODE rc(hexrc)
  REASON CODE reas(hexreas) DATE date TIME time.

  BLG25018I sessid THE HLAPI COMPLETED transcount TRANSACTIONS.
  ```

- HIGH_MEMORY must contain the character value YES, which specifies that the HLAPI may return output, message, and error PDBs in memory that was obtained above the 16MB address range. If you specify any other value, these PDBs are always returned in memory obtained below the 16MB address range. If you are using the HLAPI through a remote client, do not use this PDB, because the value YES is always assumed.
- BYPASS_PANEL_PROCESSING must contain the character value **YES** to indicate that no panels (other than those for the delete transaction) are used in record processing. If you specify any other value, the HLAPI performs panel processing. If you specify a value of YES, you must also use data model records if you are using file processing transactions (create, update, or add record relation).
- DATE_FORMAT must contain a supported date format. All dates passed between your application and the API will be in this format. Dates you pass into the API in this format will be converted to the primary date format of the database before being processed.
- MULTIPLE_RESPONSE_FORMAT indicates whether spaces can be used to separate responses in multiple response fields. This control PDB only applies to the process of multiple responses. Specify the character value PHRASE if you want to permit spaces to separate responses in a multiple response field. Specify the character value SEPARATOR (or any value other than PHRASE) to cause the value specified in SEPARATOR_CHARACTER to separate words of a multiple response.

HICAINPP (INPUT)
Initialize to zeros.

HICAOUTP (OUTPUT)
Initialize to zeros.

HICAMSGP (MESSAGES)
Initialize to zeros.

HICAERRP (ERROR CODES)
Initialize to zeros.

HICASTPA
Initialize to zeros.

Table 50 shows the initialize Tivoli Information Management for z/OS (HL01) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL01. This style reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields, and their parameters, see "HLAPI Structures" on page 216.

Table 50. HLAPI Transaction HL01. Initialize Tivoli Information Management for z/OS

Step	Program	Action
1	Application	■ Establishes linkage to module BLGYHLPI and saves its address ■ Gets storage for a HICA ■ Sets fields as follows: • HICAACRO=HICA • HICALENG=length of HICA • HICASTPA=0000 • HICAENVP=0000 • HICAINPP=0000 • HICAOUTP=0000 • HICAMSGP=0000 • HICAERRP=0000 • HICACTLP (pointer to first control PDB) The following PDBs are required: – TRANSACTION_ID=HL01 – APPLICATION_ID = the ID of your application – SESSION_MEMBER = the load library session parameter member name – PRIVILEGE_CLASS = the privilege class name The following PDBs are optional: – TABLE_COUNT = number of alias tables, non-inquiry data tables and related pattern tables in session – CLASS_COUNT = number of privilege class records in session – APIMSG_OPTION = destination of LLAPI message output – HLIMSG_OPTION = destination of HLAPI message output – TIMEOUT_INTERVAL = transaction processing time in seconds – SPOOL_INTERVAL = number of minutes that activity logs are spooled – DATABASE_ID = ID number of database used – DEFAULT_OPTION = specifies how the API processes create record default responses – DEFAULT_DATA_STORAGE_SIZE=additional storage for create record default response data in bytes – HIGH_MEMORY=YES to use memory above the 16MB address range – HLAPILOG_ID = session ID in HLAPI log messages – BYPASS_PANEL_PROCESSING = specifies whether you want to use panel processing or bypass panel processing – DATE_FORMAT = format to use for passing dates between your application and the API – MULTIPLE_RESPONSE_FORMAT = format to use for passing multiple response field data between your application and the API. ■ BLGYHLPI(*HICA*)

Table 50. HLAPI Transaction HL01 (continued). Initialize Tivoli Information Management for z/OS

Step	Program	Action
2	Server	■ Validates HICA and PDB fields ■ Initializes HLAPI environment ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAENVP **Note:** Your application must maintain the environment block pointer until Tivoli Information Management for z/OS ends. • HICAMSGP • HICAERRP • HICASTPA ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

Terminate Tivoli Information Management for z/OS (HL02)

This transaction stops the HLAPI, the LLAPI, and the LLAPI subtask, and ends the Tivoli Information Management for z/OS environment. It also frees resources allocated by the APIs. If a timeout occurs during your session, HLAPI termination happens automatically.

The following HICA and PDB fields are used in this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The PDB TRANSACTION_ID is required and must contain the 4-character transaction code of HL02.

HICAINPP (INPUT)
Initialize to zeros.

HICAOUTP (OUTPUT)
Contains the value stored by the HLAPI from the previous transaction

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 51 on page 161 shows the terminate Tivoli Information Management for z/OS (HL02) transaction flow. In the table, symbolically named PDBs equal a value, for example, PDB TRANSACTION_ID=HL02. This style reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields, and their parameters, see “HLAPI Structures” on page 216.

Table 51. HLAPI Transaction HL02. Terminate Tivoli Information Management for z/OS

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDB is required: – TRANSACTION_ID=HL02 ■ BLGYHLPI(*HICA*)
2	Server	■ Validates HICA and PDB fields ■ Terminates HLAPI environment ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAENVP • HICAMSGP • HICAERRP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

Interface Service Transactions

These transactions provide unique services to your application and other transactions. These services include freeing text data sets, obtaining record IDs, and checking records in and out. These interface service transactions are HL03 through HL05, and HL14 through HL16.

Obtain External Record ID (HL03)

This transaction obtains a Tivoli Information Management for z/OS external record identifier for use in Tivoli Information Management for z/OS record creation. On return to the application, the HLAPI builds an output PDB that contains an 8-character external record identifier in field PDBDATA. The PDBNAME of RNID_SYMBOL identifies this PDB. This transaction provides applications with a centralized record numbering service (of unique record identifiers) for later use in record creation. Once the HLAPI obtains the record ID, your application cannot return it to Tivoli Information Management for z/OS for reuse. The obtained ID can only be used in a create record (HL08) transaction. Do not specify record ID validation when you create a record with this record ID because record ID validation does not allow an all-numeric record ID.

Your application uses the following HICA and PDB fields in this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (pointer to first control PDB)
The PDB TRANSACTION_ID is required and must contain a 4-character transaction code of HL03.

HICAINPP (INPUT)
Initialize to zeros.

HICAOUTP (OUTPUT)
This field receives the address of the first output PDB. The PDB, named RNID_SYMBOL, contains the 8-character system-generated external record identifier returned from the HLAPI.

Note: Symbolic alias names are not allowed for this reserved symbolically named PDB. That is, you cannot define an alias for RNID_SYMBOL.

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 52 shows the obtain record ID (HL03) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL03. This style reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields, and their parameters, see "HLAPI Structures" on page 216.

Table 52. HLAPI Transaction HL03. Obtain Record ID

Step	Program	Action
1	Application	■ Sets the fields as follows: • HICAINPP=0000 • HICACTLP (pointer to first control PDB) The following PDB is required: – TRANSACTION_ID=HL03 • BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Gets record ID ■ Waits for completion ■ Sets following HICA fields: • HICARETC • HICAREAS • HICAOUTP • HICAMSGP • HICAERRP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAOUTP points to PDB named RNID_SYMBOL containing the record ID. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

Check Out Record (HL04)

This transaction checks out a Tivoli Information Management for z/OS record. An indicator in the record signals all users that the record is unavailable for update by any other user. This indicator does not prevent other users from attempting to access the record; it only prevents users from updating the record. Any transactions resulting in an update to the record might not access the record immediately and might have to try one or more times.

The check in record transaction (HL05) makes the record accessible for update by other users. A user might not be able to check in the record if another user is attempting to check out the record at the same time. In this case, both users must attempt to complete their transactions again.

You can direct the HLAPI to retry this transaction from 1 to 255 times before returning control to your application or wait until the record is available. See page 18 for more information.

Note: If you are using logical database partitioning, you can check out a record only if the Owning Partition of that record matches the Primary Partition of your privilege class.

In order to reduce the risk of leaving a record indefinitely in checked out status, you may wish to specify the BLX-SP parameter APICHKOUTLIM (this is described in greater detail in the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*). When a check out limit is specified, the check out record process reads the value for this parameter and performs one of the following functions:

- If the record is not already checked out, or it is checked out to a different application ID and the check out time has expired, the check out time period is added to the current clock time and stored in the record.
- If the record is already checked out to a different application ID and the check out time has not expired, an error is returned indicating that the record is in use.
- If the record is already checked out to the same application ID, then the expiration time is reset to a full check out time period and saved in the record.

The expiration time is also checked on the Update Record (HL09) transaction, the Add Record Relations (HL12) transaction, the Delete Record (HL13) transaction, and by interactive update and delete processing.

Use the following HICA fields and PDBs in this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The following PDBs are required:

- TRANSACTION_ID must contain the 4-character transaction code of HL04.
- RNID_SYMBOL must contain a 1- to 8-character identifier of the record to check out.

  Note: The 8-byte record identifier can be mixed data containing DBCS characters enclosed by a shift out (SO) and a shift in (SI) character (an SO/SI pair).

The following PDBs are optional:

- APPLICATION_ID contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses. The application ID remains in effect until changed on a subsequent transaction. If APISECURITY=ON is specified in your BLX-SP startup parameters, you must ensure that the MVS user IDs running your application are allowed to use this application ID. “API Security” on page 287 contains additional information regarding API security.
- PRIVILEGE_CLASS contains a 1- to 8-byte uppercase startup privilege class name that the API passes to Tivoli Information Management for z/OS. This privilege class

remains in effect until changed on a subsequent transaction. A privilege class name can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana. The application ID used must be an eligible user of this privilege class.

HICAINPP (INPUT)
Initialize to zeros.

HICAOUTP (OUTPUT)
Contains the value previously stored by the HLAPI.

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 53 shows the check out record (HL04) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL04. This method reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields and their parameters, see "HLAPI Structures" on page 216.

Table 53. HLAPI Transaction HL04. Check Out Record

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDBs are required: – TRANSACTION_ID=HL04 – RNID_SYMBOL = record ID to be checked out The following PDBs are optional: – APPLICATION_ID = the application ID – PRIVILEGE_CLASS = the privilege class name ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Checks out specified record ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

Check In Record (HL05)

This transaction removes the checkout indicator in a record when the application or user ID stored in the record is the same as the application ID issuing the transaction request. You use

this transaction to make a checked out record accessible for update by other Tivoli Information Management for z/OS users after your update is complete. If the API returns an unavailable condition on a check in record attempt because another user is attempting to update the record at the same time your application is attempting to check the record in, your application should restart the check in record transaction until it succeeds. You can direct the HLAPI (via the LLAPI) to retry this transaction from 1 to 255 times before returning control to your application or wait until the record is available. See page 18 for more information.

Note: If you are using logical database partitioning, you can check in a record only if the Owning Partition of that record matches the Primary Partition of your privilege class.

The following HICA fields and PDBs are used in this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The following PDBs are required:

- TRANSACTION_ID must contain the 4-character transaction code HL05.
- RNID_SYMBOL must contain a 1- to 8-character identifier of the record to check in. The 8-byte record identifier can be mixed data containing DBCS characters enclosed by an SO/SI pair.

The following PDBs are optional:

- APPLICATION_ID contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses. The application ID remains in effect until changed on a subsequent transaction. If APISECURITY=ON is specified in your BLX-SP startup parameters, you must ensure that the MVS user IDs running your application are allowed to use this application ID. “API Security” on page 287 contains additional information regarding API security.
- PRIVILEGE_CLASS contains a 1- to 8-byte uppercase startup privilege class name that the API passes to Tivoli Information Management for z/OS. This privilege class remains in effect until changed on a subsequent transaction. A privilege class name can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana. The application ID used must be an eligible user of this privilege class.

HICAINPP (INPUT)
Initialize to zeros.

HICAOUTP (OUTPUT)
Contains the value previously stored by the HLAPI.

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 54 on page 166 shows the check in record (HL05) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL05. This method reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is

the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields, and their parameters, see “HLAPI Structures” on page 216.

Table 54. HLAPI Transaction HL05. Check In Record

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDBs are required: – TRANSACTION_ID=HL05 – RNID_SYMBOL = the ID of the record being checked in The following PDBs are optional: – APPLICATION_ID = the application ID – PRIVILEGE_CLASS = the privilege class name ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Checks in specified record ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

Start User TSP or TSX (HL14)

This transaction starts a user Terminal Simulator Panel (TSP) or Terminal Simulator Exec (TSX) and passes a parameter to it. Your application can specify the name of a TSP or TSX to invoke or else you can specify the name of the TSP or TSX in BLGAPI00 (if you are using panel processing) or BLGAPIDI (if you are using bypass panel processing). See LLAPI transaction T111, “Start User TSP or TSX (T111)” on page 52 for more information.

The APIs impose certain product command restrictions. For this reason, existing user-written TSPs or TSXs might not run correctly when started from the HLAPI. For more information about these restrictions, see “Command Limitations” on page 24.

The TSP or TSX should always end by resuming any suspended sessions and by performing an `;INITIALIZE` to reset the environment in which the API is running.

The application can specify the PDB TSP_NAME to define a TSP or TSX to be invoked, and the PDB USER_PARAMETER_DATA to define a string to pass to the TSP (using the variable data area) or TSX (as an argument). The maximum length of the string is 255. You can only pass a parameter string to a TSP or TSX that your application specified in the input PDB TSP_NAME. You can also pass effectively an unlimited amount of data to an invoked TSX by specifying input PDBs. The invoked TSX can use TSX control line GETAPIDATA to access the specified data. An invoked TSX can return data to the calling application by using the TSX control line SETAPIDATA. Data is returned to the calling application in the

form of output PDBs. The *Tivoli Information Management for z/OS Terminal Simulator Guide and Reference* contains additional information on the GETAPIDATA and SETAPIDATA control lines.

You can use user exit BLGYAPSR to set a reason code in the HICAREAS field when your TSP or TSX completes. You must use reason codes 1000 to 9999 for user definition. If BLGYAPSR sets a reason code, the associated return code is 12.

You can also set any Return and Reason Code value that you want. Use SETAPIDATA to return output values for HICARETC and HICAREAS. If no other errors occur running the HL14 transaction and you set both HICARETC and HICAREAS to non-zero values, the HLAPI will set HICARETC and HICAREAS to the values you specified.

You use the following HICA fields and PDBs for this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The following PDB is required:

- TRANSACTION_ID must contain the 4-character transaction code of HL14.

The following PDBs are optional:

- APPLICATION_ID contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses. The application ID remains in effect until changed on a subsequent transaction. If APISECURITY=ON is specified in your BLX-SP startup parameters, you must ensure that the MVS user IDs running your application are allowed to use this application ID. “API Security” on page 287 contains additional information regarding API security.
- PRIVILEGE_CLASS contains a 1- to 8-byte uppercase startup privilege class name that the API passes to Tivoli Information Management for z/OS. This privilege class remains in effect until changed on a subsequent transaction. A privilege class name can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana. The application ID used must be an eligible user of this privilege class.

HICAINPP (INPUT)

- TSP_NAME contains the name of a TSP or TSX to run. A parameter can be passed to the TSP (using the variable data area) or TSX (as an argument) by specifying the string in an input PDB USER_PARAMETER_DATA.
- USER_PARAMETER_DATA contains a 1– to 255–byte character string passed to the TSP or TSX named in TSP_NAME. The value specified for the PDB USER_PARAMETER_DATA is ignored if TSP_NAME is not specified. If USER_PARAMETER_DATA is not specified, the pointer contained in USER_PARAMETER is put into TSCAUPTR.
- USER_PARAMETER contains a pointer to a user-defined area. If both USER_PARAMETER_DATA and USER_PARAMETER are specified, USER_PARAMETER is ignored.

You can specify the input PDBs to be accessed by an invoked TSX. Any input PDBs that your application specifies can be accessed by an invoked TSX by using the TSX

control line GETAPIDATA. See the *Tivoli Information Management for z/OS Terminal Simulator Guide and Reference* for additional information on GETAPIDATA.

HICAOUTP (OUTPUT)
You can return output PDBs by using the TSX control line SETAPIDATA. The *Tivoli Information Management for z/OS Terminal Simulator Guide and Reference* contains additional information on this control line.

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 55 shows the start user TSP (HL14) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL14. This method reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields and parameters, see "HLAPI Structures" on page 216.

Table 55. HLAPI Transaction HL14. Start User TSP

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDB is required: – TRANSACTION_ID=HL14 • HICAINPP = address of first input PDB The following PDBs define the name of the TSX to invoke and parameter data: – TSP_NAME – USER_PARAMETER_DATA – USER_PARAMETER The following PDBs are optional: ■ APPLICATION_ID = the application ID ■ PRIVILEGE_CLASS = the privilege class name ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Starts user TSP ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAOUTP contains pointer to output PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

Free Text Data Set (HL15)

This transaction frees HLAPI-allocated text data sets. Freeing releases all HLAPI resources associated with each data set.

If you intend to have your application keep the data set after a retrieve (HL06) transaction, then code your application to perform a Free Text Data Set transaction (HL15) right after the retrieve transaction finishes. The HLAPI does not reuse the data set on subsequent transactions.

If you perform a Free Text Data Set transaction (HL15) to free a data set, you cannot then perform a Delete Text Data Set transaction (HL16) to free that data set. You must use *TSO* to delete a data set that you freed using the Free Text Data Set transaction (HL15) transaction.

Use the following HICA fields and PDBs for this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The PDB TRANSACTION_ID is required and must contain the 4-character transaction code of HL15.

HICAINPP (INPUT)
The address of the first text data set PDB. Each PDB is named TEXT_DDNAME, and its PDBDATA value specifies a unique text DDNAME identifying the data set to be freed. Each DDNAME must be the complete 8-byte DDNAME, not the DDNAME prefix used with transactions that create the data set. For example, you run the retrieve transaction (HL06) with a TEXT_DDNAME PDB value of MYDATA, generating DDNAMEs of MYDATA01, MYDATA02 and MYDATA03. To free these DDNAMEs, specify three PDBs, each with a different full 8-byte DDNAME, not the 6-byte value MYDATA used to create them.

HICAOUTP (OUTPUT)
Contains the value previously stored by the HLAPI.

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 56 on page 170 shows the free text data set (HL15) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL15. This method reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields and their parameters, see “HLAPI Structures” on page 216.

Table 56. HLAPI Transaction HL15. Free Text Data Set

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDB is required: – TRANSACTION_ID=HL15 • HICAINPP = the address of the first TEXT_DDNAME PDB ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Frees HLAPI data set resources ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

Delete Text Data Set (HL16)

This transaction deletes one or more HLAPI-allocated text data sets. This transaction also releases all HLAPI resources associated with each data set. That is, it performs the function of transaction HL15.

If you intend to have your application delete the data set after a retrieve (HL06) transaction, then code your application to perform a Delete Text Data Set (HL16) transaction right after the retrieve transaction finishes.

If you perform a Free Text Data Set transaction (HL15) to free a data set, you cannot then perform a Delete Text Data Set transaction (HL16) to free that data set. You must use TSO to delete a data set that you freed using the Free Text Data Set transaction (HL15) transaction.

Note: Be sure to process the data in the data set before deleting it.

Use the following HICA fields and PDBs for this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The PDB TRANSACTION_ID is required and must contain the 4-character transaction code of HL16.

HICAINPP (INPUT)
The address of the first text data set PDB. Each PDB is named TEXT_DDNAME, and its data value specifies a unique text DDNAME for the data set you want deleted. Each DDNAME must be the complete 8-byte DDNAME, not the DDNAME prefix used with transactions that create the data set. For example, you run the retrieve transaction (HL06) with a TEXT_DDNAME PDB value of MYDATA, generating DDNAMEs of

MYDATA01, MYDATA02, and MYDATA03. To delete these DDNAMEs, specify three PDBs, each with a different full 8-byte DDNAME, not the 6-byte value MYDATA used to create them.

HICAOUTP (OUTPUT)
Contains the value previously stored by the HLAPI.

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 57 shows the delete text data set (HL16) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL16. This style reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields and their parameters, see "HLAPI Structures" on page 216.

Table 57. HLAPI Transaction HL16. Delete Text Data Set

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDB is required: – TRANSACTION_ID=HL16 • HICAINPP = the address of the first TEXT_DDNAME PDB ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Frees HLAPI resources and frees text data set ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

Database Access Transactions

Use this group of transactions to retrieve, create, update, inquire about, add record relation to, and delete records in the Tivoli Information Management for z/OS database. The database access transactions are HL06, HL08, HL09, HL11, HL12, and HL13.

Retrieve Record (HL06)

This transaction retrieves specific information or all information from a Tivoli Information Management for z/OS record in the database. The HLAPI provides you with a retrieve list mechanism that lets you extract specific fields from the record rather than all record fields.

You can only extract data from fields in the record that match fields in the PIDT you specify. The HLAPI returns data in an output PDB chain with one PDB allocated by the HLAPI for each data item.

Note: If you are using logical database partitioning, you can retrieve a record only if the Owning Partition of that record matches the Primary Partition of your privilege class.

If you request text retrieval and the HLAPI allocates data sets, you must issue a free (HL15) or delete (HL16) transaction immediately after processing the returned text and before issuing another retrieve (HL06) transaction.

Use the following HICA fields and PDBs for this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The following PDBs are required:

- TRANSACTION_ID must contain the 4-byte character transaction code of HL06.
- RNID_SYMBOL must contain a 1- to 8-character identifier of the record to retrieve.

  Note: The 8-byte record identifier can be mixed data containing DBCS characters enclosed by a shift out (SO) and a shift in (SI) character (an SO/SI pair).
- Either PIDT_NAME or DATA_VIEW_NAME so that the HLAPI can perform data view processing. A static PIDT table or a data view record defines the view of the record the HLAPI processes. You can define just the fields that your application requires. See "Field Validation Using the Field Validation Module BLGPPFVM" on page 279 for additional information. If both PIDT_NAME and DATA_VIEW_NAME are specified, the HLAPI ignores PIDT_NAME.
  - PIDT_NAME must contain the alias or member name of the static retrieve PIDT table the HLAPI uses in processing the transaction. Member names are 1 to 7 uppercase characters long. Alias names are 32 uppercase characters long. You create static PIDTs by using the Table Build Utility.
  - DATA_VIEW_NAME contains a character field that specifies a data view name either as an alias or data view record ID. You must specify an alias name as a 32-character left-justified field exactly as it appears in a PALT. You specify a data view name as a 1- to 8-character name. A PIDT is generated from the data view record and associated data attribute and validation records.

  Note: All PIDTs and related PIPTs can be maintained in storage to improve performance. This can be especially important if you are using data view records, as it can take a significant amount of time to generate the PIDT from the data model records. The composition of the static PIDT or data view record can affect applications that also provide an input list in that applications can request data that the static PIDT or data view record do not define. Fields defined in the static PIDT or data view record are the only fields available for retrieval.

The following PDBs are optional:

- APPLICATION_ID contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses. The application ID remains in effect until changed on a subsequent transaction. If APISECURITY=ON is specified in your BLX-SP startup parameters, you must ensure that the MVS user IDs running your application are allowed to use this application ID. “API Security” on page 287 contains additional information regarding API security.
- PRIVILEGE_CLASS contains a 1- to 8-byte uppercase startup privilege class name that the API passes to Tivoli Information Management for z/OS. This privilege class remains in effect until changed on a subsequent transaction. A privilege class name can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana. The application ID used must be an eligible user of this privilege class.
- TEXT_OPTION must contain the character value YES, which indicates that the HLAPI processes all text items. If you omit this PDB, or if it contains any value other than YES, the HLAPI bypasses text processing, and no error occurs.
- TEXT_MEDIUM must contain the character B or D that specifies which of two storage mediums you want to use for text data storage.
  - If you want text stored in a data set, specify the character D for this field. When you specify TEXT_MEDIUM as D, then TEXT_DDNAME is the only valid PDB associated with TEXT_MEDIUM. If the value in HICAINPP is nonzero, text processing is not performed, and warning codes are returned when the transaction completes.
  - If you want text stored in a storage buffer, specify the character B for this field. When you specify TEXT_MEDIUM as B, then all PDBs whose names start with TEXT except TEXT_DDNAME are associated with TEXT_MEDIUM.
  - If you omit this value or specify any character other than D or B, then the HLAPI assumes the value of D.

  This PDB is ignored if TEXT_OPTION is not YES.
- TEXT_UNITS must contain a 4-byte fixed value that specifies the maximum number of text units (lines) that the API can store in the response buffer for each text type.

  You use this PDB to limit the amount of text for any one text type. This PDB applies only to the retrieve record (HL06) transaction. The API processes this PDB only if TEXT_MEDIUM is B and TEXT_OPTION is YES. You use this PDB with the TEXT_AREA parameter. If you omit this PDB or it is zero and you process text in the response buffer, then a default value of 60 units (lines) is assumed.

  This PDB is ignored if TEXT_STREAM is YES.
- TEXT_WIDTH must contain a 4-byte fixed value that specifies the maximum width of a text unit (line) that the API stores in the response buffer. You use this PDB only with record retrieval transactions and when TEXT_MEDIUM is B and TEXT_OPTION is YES. Text width can be any value between 1 and 132. If TEXT_WIDTH is zero, omitted, or greater than 132, and you choose to process text in the response buffer, then a default value of 60 is assumed. If you are retrieving audit data, the API does not process it as part of the text, but the API appends audit data to the end of the text. Therefore, the amount of data in each returned line equals TEXT_WIDTH (in bytes) plus 36 bytes for audit data. If you are not retrieving audit data, the amount of data in each returned line equals TEXT_WIDTH.

This PDB is ignored if TEXT_STREAM is YES.

- TEXT_AREA can contain the character B or T that specifies whether the bottom block or top block of text data is stored in the PDB when the number of text units available exceeds the amount specified by TEXT_UNITS. This PDB applies only to the retrieve record (HL06) transaction and is processed only when TEXT_MEDIUM is B and TEXT_OPTION is YES.
  - B specifies that the HLAPI stores the bottom block of text.
  - T specifies that the HLAPI stores the top block of text.

  If you omit this parameter, or if it contains a value other than T or B, then the API assumes a default value B. Use this parameter with the TEXT_UNITS PDB.

  This PDB is ignored if TEXT_STREAM is YES.

- TEXT_DDNAME that you must specify if your application wants to assign user-defined DDNAME prefixes for text data sets. The data value for this PDB must be a 6-byte uppercase character value (DDNAME prefix) to which the HLAPI appends the numbers 01-99. That is, only 99 text DDNAMES are available for an individual Tivoli Information Management for z/OS record. The HLAPI returns a data set having a different DDNAME for each text type in the record. If you omit this PDB, the HLAPI assigns a default DDNAME value of BLGTXT*nn* incrementing the *nn* value each time it allocates a text data set for a unique data type in an individual Tivoli Information Management for z/OS record.

  Note: If you use the same DDNAME prefix for each record retrieved, your application must process the text immediately after transaction completion and issue a free (HL15) or delete (HL16) data set transaction, or subsequent text returns are unpredictable.

  The API processes this PDB only when TEXT_MEDIUM is not B and TEXT_OPTION is YES.

- TEXT_STREAM determines how text is stored in the response buffer. If TEXT_STREAM is omitted or contains any value other than YES, text is stored as a series of fixed-width lines. TEXT_WIDTH specifies the width of each line and TEXT_UNITS specifies the number of lines.

  If TEXT_STREAM is YES, text is stored as a continuous stream of data. Carriage return / line feed characters (EBCDIC X'0D25') characters are stored in the response buffer after each text line is read from the record. If a text line is an extension, no carriage return / line feed is stored in the response buffer after the line. See the description of TEXT_STREAM in “Create Record (HL08)” on page 178 for more information on text line extensions.

  If text is being retrieved by a workstation application, the EBCDIC carriage return / line feed characters will be translated to the appropriate ASCII characters.

  If TEXT_STREAM is YES, TEXT_OPTION must be YES, TEXT_AUDIT_OPTION must be NO, and TEXT_MEDIUM must be B.

- TEXT_AUDIT_OPTION must contain the character value NO, which indicates that the HLAPI should not return text audit data. If you omit this PDB, or if it contains any value other than NO, the HLAPI returns text audit data. This PDB is ignored if TEXT_OPTION is not YES.

- A PDB named ALIAS_TABLE containing a left-justified 1- to 8- uppercase character alias table name used for this transaction. If you omit this PDB or it does not have a value, the HLAPI does not perform any alias table processing. See "Alias Tables" on page 238 for more information on alias processing.
- HISTORY_DATA contains R, S, or B.
  - R specifies that the HLAPI is to return at the end of the output PDB chain all of the history data contained in the record.
  - S specifies that the HLAPI should save the PIHT retrieved with this record for later use on an update transaction. Any previously saved and unused PIHT is replaced.
  - B specifies that the HLAPI performs both functions of R and S.

  If you specify any other character or you omit this parameter, then the HLAPI does not retrieve or save history data for the record. No additional level of authority is required for this function beyond that for record retrieval. When the parameter value of this PDB is set to R or B, the PICAHIST field is set to Y for the corresponding LLAPI retrieve transaction. The HLAPI then builds an output PDB for each row of the returned PIHT. However, not all of the fields and flags defined in the PIHT are copied to the PDB.
- DATE_FORMAT must contain a supported date format. All dates passed between your application and the API will be in this format. Dates you pass into the API in this format will be converted to the primary date format of the database before being processed.

HICAINPP (INPUT)

Contains either the address of the first PDB in an input chain or zeros. When you specify an input PDB chain, the HLAPI treats it as a unique field retrieval list consisting of PDBs named RETRIEVE_ITEM. The HLAPI attempts to retrieve only those fields that have their uppercase names specified in the retrieval list made up of RETRIEVE_ITEM PDBs on the chain. However, the LLAPI retrieves all fields defined in the static PIDT or data view record and errors can be generated by fields that your application did not specifically request. Each PDBDATA field contains an item name to be retrieved. You can specify either the internal symbolic name or alias name on input chain PDBs. The API names the corresponding output PDBs using these names when using retrieval list processing.

On output, if the item contains no data, PDBCODE is set to E. If the item cannot be found in the PALT or PIDT (either static or generated), PDBCODE is set to M.

Retrieval of text data set items using retrieval list processing must not be performed because all text item data sets are allocated by the LLAPI and only requested items are passed through the HLAPI.

If you do not specify any RETRIEVE_ITEM PDBs on the input chain, the HLAPI uses the output PDB chain to return all data fields defined in the PIDT (static or generated from a data view record) and available in the record. The API names these PDBs using alias names, or s-word index or p-word index names as defined in PIDTSYMB.

HICAOUTP (OUTPUT)

The PDBs produced on this chain refer to data fields extracted from the record. See HICAINPP (INPUT) for unique field retrieval information. The API provides a PDB with the symbolic name of SEPARATOR_CHARACTER. It contains the separator character used by the HLAPI to process response data as defined in PIDTSEPC of the

specified PIDT. List entry items are separated by the separator character; multiple response items are separated by the separator character, or separated by a space if MULTIPLE_RESPONSE_FORMAT (described on page 158) was set to PHRASE at session initialization.

If you prefer to have text lines stored in data sets, the output chain PDB contains data set name information. The first 8 characters of the name information are the data set's DDNAME followed by a period. The remaining characters are the data set name qualifiers with each qualifier separated by a period.

If you elect to have text lines stored in the response buffer, the HLAPI converts them to PDBs. If TEXT_STREAM if omitted or contains any value other than YES, PDB field PDBDATW specifies the width of a text line and PDB field PDBDATL specifies the total text length. PDBDATL is a multiple of PDBDATW. If TEXT_STREAM is YES, PDBDATW and PDBDATL both equal the total text length. The text in the response buffer may contain carriage return / line feed characters that indicate the end of a text line.

When a record is retrieved, each text data set record and buffer entry includes audit data, if audit data is requested (the TEXT_AUDIT_OPTION parameter determines whether the HLAPI should return text audit data). Information on the format of audit data can be found on page 22.

The HLAPI returns visible phrase and direct-add data items to the output chain PDBDATA fields.

If history data was requested, it is returned on the output PDB chain following all of the other record data. The output PDBs that contain history data contain this information:

PDBNAME
contains the unique character string HISTORYnnnnnn where nnnnnn starts at 000001 and increases with each history data PDB on the output chain.

PDBTYPE
contains G or H.

- G specifies that this PDB comes first in a group of one or more related history data items. This is indicated by the associated PIHTSGRP row field set to Y.
- H specifies that this PDB is not the first PDB in a group of several related history data items. This is indicated by the associated PIHTSGRP row field not set to Y.

PDBDATL
a four byte length of the history data.

PDBDATA
a variable length character field containing the history data. The data may contain a prefix.

As with any PDB data returned on the output chain, the storage is freed on the next invocation of the HLAPI. Only one PIHT can be saved at a time, regardless of record type. The history saved from one record remains available for use until it is replaced by the saved PIHT of another record or until it is actually used. Subsequent transactions that do not save or use history data, retrieve or otherwise, have no effect on the saved history.

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 58 shows the retrieve record (HL06) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL06. This method reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields and parameters, see "HLAPI Structures" on page 216.

Table 58. HLAPI Transaction HL06. Retrieve Record

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDBs are required: – TRANSACTION_ID=HL06 – RNID_SYMBOL = ID of record to be retrieved – Either of the following: ■ PIDT_NAME specifies the name or alias of the static retrieve PIDT table the HLAPI uses in processing the transaction. ■ DATA_VIEW_NAME specifies a data view name or alias of a data view record ID that the HLAPI uses in processing the transaction. The following PDBs are optional: – APPLICATION_ID = the application ID – PRIVILEGE_CLASS = the privilege class name – ALIAS_TABLE = alias table name for this transaction – TEXT_OPTION=YES to enable text processing – TEXT_AUDIT_OPTION=NO to not return text audit data – DATE_FORMAT = format to use for passing dates between your application and the API The following PDBs (data set text processing) are optional: ■ TEXT_MEDIUM=D ■ TEXT_DDNAME = user defined 6-character DDNAME prefix The following PDBs (buffer text processing) are optional: ■ TEXT_MEDIUM=B ■ TEXT_UNITS = maximum text lines in buffer ■ TEXT_WIDTH = maximum width of a text line in buffer ■ TEXT_AREA=B for bottom text block, T for top text block ■ HISTORY_DATA specifies whether the HLAPI is to return at the end of the output PDB chain all of the history data contained in the record or whether the HLAPI should save the PIHT retrieved with this record for later use on an update transaction. The following PDBs (text stream buffer processing) are optional: ■ TEXT_MEDIUM=B ■ TEXT_STREAM = YES • HICAINPP = zeros or the address of the first RETRIEVE_ITEM PDB (containing the name of a field to retrieve) on the input chain **Note:** If field contains zeros (no PDB address specified), the HLAPI retrieves all fields defined in the PIDT for the record type. ■ BLGYHLPI(*HICA*).

Table 58. HLAPI Transaction HL06 (continued). Retrieve Record

Step	Program	Action
2	Server	■ Validates HICA and PDB fields ■ Retrieves record ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP • HICAOUTP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. • HICAOUTP contains pointer to retrieval results (output) chain. • PDBCODE in input PDBs to determine if no data was returned in the field or the field was not defined in the PALT or PIDT. ■ Continues processing.

Create Record (HL08)

This transaction creates records in the Tivoli Information Management for z/OS database. The HLAPI accepts data from the input PDBs and builds a Tivoli Information Management for z/OS record. The record contains the input data and, if you are using panel processing, any audit data that Tivoli Information Management for z/OS normally adds to the record when the record is filed. If you are using bypass panel processing, audit data can be listed in the data view record and added to the record being created along with the input data your application specifies. Leading and trailing blanks are removed from all but text data. Do not imbed blanks in a response or include the separator character value as part of a response. You can use static PIDTs or data view records from which PIDTs are generated. If you use bypass panel processing you must use data model records. You can identify required fields for a particular record type in the static PIDT or data view record you designate for this transaction. Specify REQUIRED(Y) on the field statements to define PIDTs or define the field as required in the data view record. Use alias processing to let default data (in the alias table) be used as input for record fields you do not specify in input PDBs. Any fields you specify in input PDBs overrides default data for those fields.

You can input freeform text to the record by specifying the name of a text data set containing the text or by specifying the text itself. You can assign a user-defined or HL03-obtained record ID to a record (see the input chain pointer field HICAINPP explanation on page 180 for more information). If you do not do this, Tivoli Information Management for z/OS assigns a record ID to the created record.

If you use panel processing, TSP BLGAPI02 performs create processing. It uses some of your interactive panels to perform the create. If you use bypass panel processing, TSP BLGAPIPX performs create processing and does not use any of your interactive panels. If you plan to create records of your own type (including Tivoli Information Management for z/OS Integration Facility), have tailored your panels, or want to use existing panel

automation see “Tailoring the Application Program Interfaces” on page 289, and “Terminal Simulator Panels” on page 349 for information on interface tailoring and LLAPI create processing.

Use the following HICA fields and PDBs for this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The following PDBs are required:

- TRANSACTION_ID must contain the 4-byte character transaction code of HL08.
- Either PIDT_NAME or DATA_VIEW_NAME so that the HLAPI can perform data view processing. A static PIDT table or a data view record defines the view of the record the HLAPI processes. You can define just the fields that your application requires. See “Field Validation Using the Field Validation Module BLGPPFVM” on page 279 for additional information. If both PIDT_NAME and DATA_VIEW_NAME are specified, the HLAPI ignores PIDT_NAME.
  - PIDT_NAME must contain the alias or member name of the static retrieve PIDT table the HLAPI uses in processing the transaction. Member names are 1 to 7 uppercase characters long. Alias names are 32 uppercase characters long. You create static PIDTs by using the Table Build Utility.
  - DATA_VIEW_NAME contains a character field that specifies a data view name either as an alias or data view record ID. You must specify an alias name as a 32-character left-justified field exactly as it appears in a PALT. You specify a data view name as a 1- to 8-character name. A PIDT is generated from the data view record and associated data attribute and validation records. If you use bypass panel processing, you must specify DATA_VIEW_NAME.

  Note: All PIDTs and related PIPTs can be maintained in storage to improve performance. This can be especially important if you are using data view records, as it can take a significant amount of time to generate the PIDT from data view records.
- SEPARATOR_CHARACTER whose PDBDATA field contains the character field your application uses to separate responses for a single field (either multiple response or list item) for this create. A blank value is ignored. If you omit this parameter, the HLAPI ends the transaction with an error code.

The following PDBs are optional:

- APPLICATION_ID contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses. The application ID remains in effect until changed on a subsequent transaction. If APISECURITY=ON is specified in your BLX-SP startup parameters, you must ensure that the MVS user IDs running your application are allowed to use this application ID. “API Security” on page 287 contains additional information regarding API security.
- PRIVILEGE_CLASS contains a 1- to 8-byte uppercase startup privilege class name that the API passes to Tivoli Information Management for z/OS. This privilege class remains in effect until changed on a subsequent transaction. A privilege class name

can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana. The application ID used must be an eligible user of this privilege class.

- ALIAS_TABLE must contain the name of the alias table used for this transaction. If you omit this parameter or it does not have a value, the HLAPI does not perform any alias table processing. The field must be left justified. See “Alias Tables” on page 238 for more information about alias processing.
- DEFAULT_OPTION must contain a character field with values of ALL, REQUIRED, and NONE that specifies how the HLAPI performs default data response processing when creating the record.
  - ALL specifies that each response field in a PIDT (static or generated from a data view record) is a candidate for a default response
  - REQUIRED specifies that only required fields are candidates for default responses
  - NONE specifies that no default processing is performed.

  If you omit this field or specify it incorrectly, the HLAPI performs default option processing as it was specified in the initialize Tivoli Information Management for z/OS transaction (HL01). You can override the initial default processing option by respecifying the default option on the control chain. After the create transaction finishes, the HLAPI reverts to the initial default specification for record creation unless overridden again in subsequent transactions.
- EQUAL_SIGN_PROCESSING must contain the character value YES, which specifies that the HLAPI is to use equal sign processing. If you specify any other value, the HLAPI performs no equal sign processing. See 146 for additional information on equal sign processing.
- DATE_FORMAT must contain a supported date format. All dates passed between your application and the API will be in this format. Dates you pass into the API in this format will be converted to the primary date format of the database before being processed.
- TEXT_STREAM determines how freeform text specified in an input PDB is stored in a Tivoli Information Management for z/OS record. If TEXT_STREAM is omitted or contains any value other than YES, the text is processed as a series of fixed-width lines. PDBDATW specified the width of each line and PDBDATL specifies the total length of the text.

  If TEXT_STREAM is YES, the freeform text is processed as a continuous stream of data. This stream may contain line feed (EBCDIC X'25'), carriage return / line feed (EBCDIC X'0D25'), or newline (EBCDIC X'15') characters. When the API finds a line feed in the text stream, it stores the text following the line feed as a text line in the Tivoli Information Management for z/OS record. If there are more than 132 characters following the line feed, the first 132 characters are stored as a text line. Any remaining text, up to the next line feed or another 132 characters, is stored as a text line extension in the Tivoli Information Management for z/OS record. When the API builds text lines and text line extensions, it does not split lines in the middle of a word nor does it strip trailing blanks.

  Setting TEXT_STREAM to YES is intended to be used by applications that use the client interface to Tivoli Information Management for z/OS.

HICAINPP (INPUT)

The address of the first input PDB. The HLAPI processes PIDT table entries using

PDBs found on this chain. Include an input PDB for each data item (data, direct-add, visible phrase, and freeform text) associated with this create transaction. Set PDBNAME to an alias name or to the PIDT symbolic name of the data item, and set PDBDATA to the data value for the data item. Specify list item field instances within a single PDB using the separator character to define individual response items. Leading and trailing blanks are removed from all but freeform text data. Do not specify blanks as part of a data value.

The PIDT row corresponding to data associated with a phrase or direct-add item actually contains the data for that item. The HLAPI stores the data in the record if you include a PDB using the name of the item and a nonblank value in PDBDATA.

For example, if you want to collect s-word 0CFC with a visible phrase of REPORTER in a problem record, you can specify any nonblank value as data with the s-word (for instance, X), but only the visible phrase REPORTER is collected in the record.

If you have users who will perform interactive structured searches (that is, they use the inquiry panels), it is important to always collect the s-word associated with the summary panel for a selection.

The HLAPI validates input data when you set PDBPROC to V for each PDB whose PDBDATA you want validated. If you do not set PDBPROC to V, the HLAPI does not validate input data, and you can add incorrect data to the database. The HLAPI does not validate string, phrase, text, and direct-add items. If data fails validation, PDBCODE for the input PDB is set to V and an item is returned on the error PDB chain to indicate the reason. A list of validation codes can be found on page 236.

The HLAPI can set PDBCODE to other values. See PDBCODE on page 222 for code values returned by the API.

You can supply text data two ways:

- For text data stored in a data set, each data set name is stored in the PDBDATA field of a separate PDB.
- For text data associated with buffer processing, the values of PDBDATW and PDBDATL depend on the value of TEXT_STREAM. If TEXT_STREAM is omitted or contains any value other than YES, PDBDATW must contain the width of the text unit (line), and PDBDATL must contain the total length of the text. PDBDATL must be an even multiple of PDBDATW. PDBDATW cannot be larger than 132. If PDBDATW is zero, the PDBDATW assumes that PDBDATA contains the name of the text data set.

  If TEXT_STREAM is YES, PDBDATW and PDBDATL must both equal the total length of the text. In this case, PDBDATW can be greater than 132.

Note: You cannot use both storage buffer and data set processing when you use this transaction. You must use one or the other.

To provide a user-defined record ID or to use a record ID obtained from the Obtain Record ID (HL03) transaction, your application must provide an input PDB with the PIDT record identifier field or alias name in PDBNAME and the record ID you want set in PDBDATA. For example, if S0CCF is the PIDT s-word index of the record identifier field in a create problem record, you would assign this value to PDBNAME and you would assign the record ID you want to PDBDATA. If you use a record ID that you obtained from the Obtain Record ID (HL03) transaction, do not set PDBPROC to V to

validate the record ID. This record ID might not pass validation because assisted-entry panel validation for record IDs does not allow all-numeric record IDs.

You must specify individual list process field values with the value of the control PDB SEPARATOR_CHARACTER. You can separate words in a multiple response field with blanks if you specified MULTIPLE_RESPONSE_FORMAT=PHRASE at HL01; if you specified the character value SEPARATOR (or any value other than PHRASE), the value specified in SEPARATOR_CHARACTER is the only valid separator character.

HICAOUTP (OUTPUT)
The PDB produced on this chain is named RNID_SYMBOL and contains the 1- to 8-character external record identifier of the record created.

Note: The API does not perform alias table processing for this PDB because it uses a reserved PDB name.

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 59 on page 183 shows the create record (HL08) transaction flow. In the table, symbolically named PDBs have a value; for example, PDB TRANSACTION_ID=HL08. This method reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields, and their parameters, see “HLAPI Structures” on page 216.

Table 59. HLAPI Transaction HL08. Create Record

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to control PDB chain) The following PDBs are required: – TRANSACTION_ID=HL08 – Either of the following: ■ PIDT_NAME = the name or the alias name of the static PIDT to use in creating the record. ■ DATA_VIEW_NAME = the data view record ID or the alias of a data view record ID to use in creating the record. If you use bypass panel processing, you must use DATA_VIEW_NAME. – SEPARATOR_CHARACTER = the separator character used by the LLAPI in processing response data The following PDBs are optional: – APPLICATION_ID = the application ID – PRIVILEGE_CLASS = the privilege class name – DEFAULT_OPTION=ALL, REQUIRED, or NONE – ALIAS_TABLE = the alias table name used for this transaction – EQUAL_SIGN_PROCESSING = YES, which specifies that the HLAPI is to use equal sign processing. If you specify any other value, the HLAPI performs no equal sign processing. – DATE_FORMAT = format to use for passing dates between your application and the API – TEXT_STREAM=NO to process freeform text as fixed-width lines or TEXT_STREAM=YES to process freeform text as a continuous stream of data. • HICAINPP = the address of the first input PDB ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Creates record ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP • HICAOUTP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. • HICAOUTP contains pointer to output PDB RNID_SYMBOL that contains the ID of the created record. ■ Continues processing.

Update Record (HL09)

This transaction updates Tivoli Information Management for z/OS records in the Tivoli Information Management for z/OS database.

You prepare for data additions and changes to the record by creating an input PDB chain consisting of a PDB for each data item. You must specify at least one input data item. For each PDB, set PDBNAME to the alias name or to the PIDT symbolic name of the data item and PDBDATA to the data value for the item. The API adds data you specify to the record. It replaces existing data of the same name. Leading and trailing blanks are removed from all but text data. Do not imbed blanks in a response or include the separator character as part of a response. You can use static PIDTs or data view records from which PIDTs are generated. If you use bypass panel processing you must use data model records.

You can add freeform text to the record by specifying the name of a text data set containing the text or by specifying the text itself. You can add freeform text to that which exists in the record or replace existing freeform text.

If another application or user is attempting to update the record, the record might be unavailable. You can direct the HLAPI to either retry this transaction from 1 to 255 times before returning control to your application or wait until the record is available. See page 18 for more information.

Note: If you are using logical database partitioning, you can update a record only if the Owning Partition of that record matches the Primary Partition of your privilege class.

Checking out the record before the update ensures that no other users can update the record prior to your update. Your administrator can define a time limit for checked out records (BLX-SP parameter APICHKOUTLIM, described in the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*) so that records will not inadvertently remain indefinitely checked out if your application does not check in the record.

You can determine how you want to process lists on update. That is, you can simply update lists (this is the default), you can append new list items to existing lists, or you can replace existing lists. To specify the type of update, specify a control PDB LIST_MODE to indicate whether you want to update, append, or replace list items. The processing can be different for each update.

History Data Considerations

You can delete history data from the record. Before beginning the following sequence of actions, the Tivoli Information Management for z/OS database administrator must enable the history update feature. This is done by removing the protective branch control line in TSP BLGAPI05 (for panel processing) or BLGAPIPX (for bypass panel processing). In addition the API user (application ID) must have database administrator authority.

- The HL04 check out record transaction is used to lock a record before update, when required to maintain data integrity.
- The HL06 retrieve record transaction is used to retrieve history data along with its record data by including the control PDB HISTORY_DATA with the value of B. The history data will be saved by the HLAPI for later use, as well as being returned to the user following the record data on the output PDB chain.
- The history and record data is examined to decide if the history data is to be deleted. If so, a cutoff date is determined. If examination of the history is not required in this step, then in the previous step, use the value S for the HISTORY_DATA control PDB. This saves the history data but does not return it on the chain of output PDBs.

- If any history data is to be deleted, the HL09 update record transaction is used with the DELETE_HISTORY control PDB and the date value from the previous step. All history data recorded earlier than this date will be deleted.
- The HL05 check in record transaction is used to unlock the record, if it was locked previously.

Multiple Response Item Processing Considerations

When you input data for multiple response fields, each word of the field must be separated by the separator character specified in the control PDB named SEPARATOR_CHARACTER or by a space if MULTIPLE_RESPONSE_FORMAT (described 158) was set to PHRASE on the HL01 to allow spaces to separate multiple response words. The HLAPI locates separator characters in the SBCS data portions of responses that contain mixed data.

Field Deletion Considerations

You must explicitly identify fields to delete from the existing record.

- To delete a nonlist response item in the record, use a single separator character as a response.
- To delete freeform text, specify the REPLACE_TEXT_DATA PDB with a value of YES and specify a single separator character as the text data.

List Item Processing Considerations

When you collect list item responses, the responses must be separated by the separator character specified in the control PDB named SEPARATOR_CHARACTER. Responses do not require padding blanks. Do not append a separator character to the last response of a field. The HLAPI locates separator characters in the SBCS data portions of responses that contain mixed data.

An example of a list item using a comma separator character is `moda,modb,modc`.

An example of a skipped entry is `moda, ,modc`. (The first entry contains `moda`, and the third entry contains `modc`.)

Where list data is entered, each list response must be separated by the separator character specified in the control PDB named SEPARATOR_CHARACTER.

You can choose to update existing lists (the default), append new data to existing lists, or replace existing lists. In control PDB LIST_MODE you specify how the lists should be processed.

You can also delete data already existing in the record. To delete a response in a list of responses, specify update list processor mode (this is the default) and use 2 consecutive separator characters with the second separator character logically replacing the deleted response. A separator character in the first position of the response indicates that the first list position item is to be deleted. A trailing separator character (after the last item) indicates that the next list item of that type in the record is to be deleted. To delete an entire list, specify control PDB LIST_MODE with a value of REPLACE and a single separator character as the field data.

This example 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 PDBDATA value used to update the list, and the results of the update.:

```
List Before      PDBDATA              List After     Action Per-
Update           Value                Update         formed

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


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


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

This example shows an update transaction appending data to an existing list of routine names. For this transaction, the figure shows: the list before the transaction on the left, the response buffer segment used to append to the list, and the results of the append.

```
List Before      Response Buffer      List After     Action Performed
Append           Segment              Append

ADD              'BUILD0,BUILD1,,COPY' ADD           Append new
BUILD1                                 BUILD1       items to list
DELITEM                                DELITEM
COPY                                   COPY
--------                               --------
CHECK                                  CHECK
INIT                                   INIT
                                       BUILD0
                                       BUILD1
                                       --------
                                       COPY
```

This example shows an update transaction replacing from an existing list of routine names. For this transaction, the figure shows: the list before the transaction on the left, the response buffer segment used to replace data from the list, and the results of the replace.

```
List Before      Response Buffer      List After     Action Performed
Delete           Segment              Delete

ADD              'BUILD0,,BUILD2'     BUILD0         New data replaces
BUILD1                                --------       old data
DELITEM                               BUILD2
COPY
--------
CHECK
```

```
INIT
```

Use the following HICA fields and PDBs for this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The following PDBs are required:

- TRANSACTION_ID must contain the 4-character transaction code HL09.
- RNID_SYMBOL must contain a 1- to 8-character external record identifier of the record you want to update. A user-defined record identifier might have mixed data containing DBCS characters enclosed by an SO/SI pair. If the record identifier begins with an alphabetic character, it can be from 1 to 8 characters in length; if the record identifier begins with a numeric character, it must contain all numeric characters and must be 8 characters in length.
- Either PIDT_NAME or DATA_VIEW_NAME so that the HLAPI can perform data view processing. A static PIDT table or a data view record defines the view of the record the HLAPI processes. You can define just the fields that your application requires. See "Field Validation Using the Field Validation Module BLGPPFVM" on page 279 for additional information. If both PIDT_NAME and DATA_VIEW_NAME are specified, the HLAPI ignores PIDT_NAME.
  - PIDT_NAME must contain the alias or member name of the static update PIDT table the HLAPI uses in processing the transaction. Member names are 1 to 7 uppercase characters long. Alias names are 32 uppercase characters long. You create static PIDTs by using the Table Build Utility.
  - DATA_VIEW_NAME contains a character field that specifies a data view name either as an alias or data view record ID. You must specify an alias name as a 32-character left-justified field exactly as it appears in a PALT. You specify a data view name as a 1- to 8-character name. A PIDT is generated from the data view record and associated data attribute and validation records. If you use bypass panel processing, you must specify DATA_VIEW_NAME.

  Note: All PIDTs and related PIPTs can be maintained in storage to improve performance. This can be especially important if you are using data view records, as it can take a significant amount of time to generate the PIDT from the data model records.
- SEPARATOR_CHARACTER must contain the separator character value the HLAPI is to use to process response data for this update. A blank value is ignored. If you omit this parameter, the HLAPI ends the transaction with an error.

The following PDBs are optional:

- APPLICATION_ID contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses. The application ID remains in effect until changed on a subsequent transaction. If APISECURITY=ON is specified in your BLX-SP startup parameters, you must ensure that the MVS user IDs running your application are allowed to use this application ID. "API Security" on page 287 contains additional information regarding API security.

- PRIVILEGE_CLASS contains a 1- to 8-byte uppercase startup privilege class name that the API passes to Tivoli Information Management for z/OS. This privilege class remains in effect until changed on a subsequent transaction. A privilege class name can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana. The application ID used must be an eligible user of this privilege class.
- ALIAS_TABLE must contain the uppercase name of the alias table used for this transaction. If you omit this parameter, the HLAPI does not perform alias table processing. See "Alias Tables" on page 238 for more information about alias processing. The field must be left justified.
- EQUAL_SIGN_PROCESSING must contain the character value YES, which specifies that the HLAPI is to use equal sign processing. If you specify any other value, the HLAPI performs no equal sign processing.
- DELETE_HISTORY contains a character string specifying the date in external format of the oldest history data to be kept with the record. Any history created earlier than this date is deleted from the record. When a record is updated and the DELETE_HISTORY PDB is specified with a date value, the last saved PIHT is attached to the record and all history entries recorded before the given date are marked for deletion. PIHTs can be saved as a result of the Retrieve (HL06) transaction. The LLAPI checks to ensure that the correct PIHT (correct means that the PIHT was saved for the same record that this update is for) is attached to the record and deletes the marked entries. In the LLAPI, the history data update function is shipped disabled. Once enabled, either in TSP BLGAPI05 for panel processing or in TSP BLGAPIPX when bypassing panel processing, the user must have database administrator authority to successfully execute this transaction.

  Note: You can modify the TSPs to change the level of authority needed.
  Before data can be deleted, it must have been saved by setting the HISTORY_DATA PDB to S or B on a previous retrieve of the same record.

  Note:

  The following are limitations and/or restrictions associated with delete history processing:

  - It is not possible to delete all of the history data for a record. The history of the most recent day's activity is always kept.
  - The external date specified on the DELETE_HISTORY PDB is limited to a maximum of 32 characters.
  - There is no unique field in the history entry which contains the date when the record was changed. For the DELETE_HISTORY function to process successfully, you must journal first a data field for the record. The delete history function assumes that a prefix word beginning with DAT is used for the date and that the date field has been journaled first. If no dates are found by the delete history function, an error code is returned to indicate that a problem exists in identifying dates.
  - The dates in the history data are in external format. The currently enabled date conversion routine is used to convert them to internal format for comparison. If the conversion routine returns a non-zero return code, the data is assumed to be not a date and is skipped.

- A timestamp is kept when history data is saved. If the record is changed before the DELETE_HISTORY can be performed with the saved history, the time stamps will not match and the history will not be deleted.

- REPLACE_TEXT_DATA can contain the character value YES which indicates that any text data provided is used to replace existing text of the same type. If any other value is specified, the text data provided is appended to any existing text of the same type. When the data value of this PDB is set to YES, the PICATXTR field is set to Y for the corresponding LLAPI update transaction. This causes the existing text to be replaced by any input text with the same type. If the input data consists of a single separator character, the existing text data is deleted. For deleting freeform text using buffer processing, both PDBDATL and PDBDATW must be set to the value 1.
- DATE_FORMAT must contain a supported date format. All dates passed between your application and the API will be in this format. Dates you pass into the API in this format will be converted to the primary date format of the database before being processed.
- LIST_MODE can be used to indicate how you want to process lists on update. You can specify UPDATE to update lists, specify APPEND to append new list items to existing list, or specify REPLACE to replace existing lists. If a value is not specified, the default is UPDATE.
- TEXT_STREAM determines how freeform text specified in an input PDB is stored in a Tivoli Information Management for z/OS record. If TEXT_STREAM is omitted or contains any value other than YES, the text is processed as a series of fixed-width lines. PDBDATW specified the width of each line and PDBDATL specifies the total length of the text.

  If TEXT_STREAM is YES, the freeform text is processed as a continuous stream of data. This stream may contain line feed (EBCDIC X'25'), carriage return / line feed (EBCDIC X'0D25'), or new line (EBCDIC X'15') characters. When the API finds a line feed in the text stream, it stores the text following the line feed as a text line in the Tivoli Information Management for z/OS record. If there are more than 132 characters following the line feed, the first 132 characters are stored as a text line. Any remaining text, up to the next line feed or another 132 characters, is stored as a text line extension in the Tivoli Information Management for z/OS record. When the API builds text lines and text line extensions, it does not split lines in the middle of a word nor does it strip trailing blanks.

  Setting TEXT_STREAM to YES is intended to be used by applications that use the client interface to Tivoli Information Management for z/OS.

HICAINPP (INPUT)

The address of the first input PDB. The HLAPI processes PIDT table entries using PDBs found on this chain. You include an input PDB for each data item (data, direct-add, visible phrase, and freeform text) associated with this update transaction. Set PDBNAME to an alias name or to the PIDT symbolic name of the data item, and set PDBDATA to the data value for the data item. You specify list item field instances within a single PDB using the separator character to define individual response items. Leading and trailing blanks are removed from all but freeform text data. Do not include blanks as part of a data value. See “Multiple or List Data Item Processing Considerations” on page 76 for more information.

The API collects phrase and direct data items if you include a PDB using the name of the phrase or direct-add item and a nonblank value in PDBDATA. The PIDT row corresponding to data associated with a phrase or direct-add item actually contains the data for that item.

The HLAPI validates input data when you set PDBPROC to V for each PDB whose PDBDATA you want validated. If you do not set PDBPROC to V, the HLAPI does not validate input data and you can add incorrect data to the database. The HLAPI does not validate string, phrase, text, and direct-add items. If data fails validation, PDBCODE for the input PDB is set to V and an item is returned on the error PDB chain to indicate the reason. A list of validation codes can be found on page 236.

The HLAPI can set PDBCODE to other values. See PDBCODE, page 222, for code values returned by the API.

You can supply text data two ways:

- For text data stored in a data set, store each data set name in the PDBDATA field of a separate PDB. Set PDBDATL to the length of the data set name, and set PDBDATW to zero.
- For text data associated with buffer processing, the values of PDBDATW and PDBDATL depend on the value of TEXT_STREAM. If TEXT_STREAM is omitted or contains any value other than YES, PDBDATW must contain the width of the text unit (line), and PDBDATL must contain the total length of the text. PDBDATL must be an even multiple of PDBDATW. PDBDATW cannot be larger than 132. If PDBDATW is zero, the HLAPI assumes that PDBDATA contains the name of the text data set.

  If TEXT_STREAM is YES, PDBDATW and PDBDATL must both equal the total length of the text. In this case, PDBDATW can be greater than 132.

Note: You cannot use both storage buffer and data set processing when you use this transaction. You must use one or the other.

HICAOUTP (OUTPUT)
Contains the value previously stored by the HLAPI.

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 60 on page 191 shows the update record (HL09) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL09. This method reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields and their parameters, see “HLAPI Structures” on page 216.

Table 60. HLAPI Transaction HL09. Update Record

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDBs are required: – TRANSACTION_ID=HL09 – RNID_SYMBOL = the ID of the record to be updated – Either of the following: ■ PIDT_NAME = the name or alias of the static PIDT table the HLAPI uses in processing the transaction. ■ DATA_VIEW_NAME = the data view record ID or alias of the data view record ID that the HLAPI uses in processing the transaction. If you use bypass panel processing, you must use DATA_VIEW_NAME. – SEPARATOR_CHARACTER = the character used by the HLAPI in processing response data. The following PDBs are optional: – APPLICATION_ID = the application ID – PRIVILEGE_CLASS = the privilege class name – ALIAS_TABLE = the name of the alias table used for this transaction – EQUAL_SIGN_PROCESSING = YES, which specifies that the HLAPI is to use equal sign processing. If you specify any other value, the HLAPI performs no equal sign processing. – DELETE_HISTORY = date of oldest history data to be kept with the record – REPLACE_TEXT_DATA = YES if new freeform text is to replace existing freeform text – DATE_FORMAT = format to use for passing dates between your application and the API – LIST_MODE = UPDATE or APPEND or DELETE to indicate how you want to process lists on update – TEXT_STREAM = NO to process freeform text as fixed-width lines, YES to process freeform text as a continuous stream of data • HICAINPP = the address of the first input PDB ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Updates the record ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

Change Record Approval (HL10)

This transaction provides a means to approve or reject a change record. By using this transaction, you can pass approvals from another change management product or application,

or from a Web application into Tivoli Information Management for z/OS. This is similar to the process to approve or reject changes that you can do interactively; additional information on the interactive process to perform this function can be found in *Tivoli Information Management for z/OS Problem, Change, and Configuration Management*. A specified change record is updated as follows:

- If approval status is specified as ACCEPT, the current privilege class in the list of approvers within the change record is marked as "approval accepted."
- If the status is specified as REJECT or anything other than ACCEPT, the current privilege class in the approver list is marked as "approval rejected."
- When one approver rejects the change, the change record is marked as "rejected."
- When all of the approvers on the list have accepted the change, the change records is marked as "accepted."
- Before the change record is marked "accepted" or "rejected," it is in the "approval pending" status.

Note: If data attribute records are used as direct-add fields when creating change records, then normal file processing is not performed for change records when change approval processing is being performed. That is, if ALL of these five direct-adds—DATE/, TIME/, CLAE/, DATM/, and TIMM/—are changed to data attribute records, then data modified, time modified, and user ID are not saved in the record.

The following HICA fields and PDBs are used in this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The following PDBs are required:

- TRANSACTION_ID must contain the 4-character transaction code HL10.
- RNID_SYMBOL must contain a 1- to 8-character identifier of the change approval record. The 8-byte record identifier can be mixed data containing DBCS characters enclosed by an SO/SI pair.
- Either PIDT_NAME or DATA_VIEW_NAME so that the HLAPI can perform data view processing. A static PIDT table or a data view record defines the view of the record the HLAPI processes. This is only used to obtain the authorization code that applies to change record display. If both PIDT_NAME and DATA_VIEW_NAME are specified, the HLAPI ignores PIDT_NAME.
  - PIDT_NAME contains the alias or member name of a static PIDT used to retrieve change records. A data view record can be used in place of a static PIDT. If you choose to use a data view record, provide its name in the PDB DATA_VIEW_RECORD. Ensure that the data view record has the authority to retrieve or display change records.
  - DATA_VIEW_NAME contains a character field that specifies a data view name either as an alias or data view record ID. You must specify an alias name as a 32-character left-justified field exactly as it appears in a PALT. You specify a data view name as a 1- to 8-character name. A PIDT is generated from the data

view record and associated data attribute and validation records. If you use bypass panel processing, you must specify DATA_VIEW_NAME.

Note: All PIDTs and related PIPTs can be maintained in storage to improve performance. This can be especially important if you are using data view records, as it can take a significant amount of time to generate the PIDT from the data model records.

The following PDBs are optional:

- APPLICATION_ID contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses. The application ID remains in effect until changed on a subsequent transaction. If APISECURITY=ON is specified in your BLX-SP startup parameters, you must ensure that the MVS user IDs running your application are allowed to use this application ID. "API Security" on page 287 contains additional information regarding API security.
- PRIVILEGE_CLASS contains a 1- to 8-byte uppercase privilege class name that the API passes to Tivoli Information Management for z/OS and defines the privilege class accepting or rejecting the change. This privilege class remains in effect until changed on a subsequent transaction. A privilege class name can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana. The application ID used must be an eligible user of this privilege class. If no privilege class is set with this transaction, then the privilege class currently in effect is used.
- APPROVAL_STATUS must be set to ACCEPT to specify an accepted approval status; if no approval status is specified or if the status is not "accepted," the default is to reject the approval of the change record.
- ALIAS_TABLE can contain the name of an alias table. If no alias table is set with this transaction, then the alias table currently in effect is used. The alias table is not used with the value of the APPROVAL_STATUS PDB.
- APPROVER contains a 1- to 8-character uppercase approver name that defines the approver accepting or rejecting the change and passes this name to Tivoli Information Management for z/OS. If this PDB is specified, this value is used in place of the privilege class name that is specified on the PRIVILEGE_CLASS PDB. This approver name can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana.

Table 61 on page 194 shows the change record approval (HL10) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL10. This method reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields, and their parameters, see "HLAPI Structures" on page 216.

Table 61. HLAPI Transaction HL10. Change Record Approval

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDBs are required: – TRANSACTION_ID=HL10 – RNID_SYMBOL = the ID of the change record The following PDBs are optional: – APPLICATION_ID = the application ID – PRIVILEGE_CLASS = the privilege class name – PIDT_NAME = name of static PIDT – APPROVAL_STATUS = ACCEPT (to approve change record) – ALIAS_TABLE = name of an alias table ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Validates approval status ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

Record Inquiry (HL11)

This transaction performs a search of the Tivoli Information Management for z/OS database. It converts data parameters specified on the INPUT PDB chain to PIDT and PIAT arguments processed by the LLAPI. The HLAPI returns results of the search to an output PDB chain.

Two categories of inquiry parameters are structured argument lists and freeform argument lists. Your application can use each type independently or combined. If you choose combined argument processing, the HLAPI appends the freeform argument list to the structured argument list regardless of the order in which you specify them. Structured arguments simulate interactive quick-search dialog field responses, while freeform arguments simulate interactive freeform arguments. Terminal session command line arguments and those arguments created using the ARG command in a terminal session are examples of freeform arguments.

To increase your ability to eliminate unwanted records from the results of freeform searches, you can use parentheses within freeform search arguments to specify the order in which arguments should be evaluated. Arguments placed within parentheses will be evaluated first. The parentheses can adjoin the arguments or be separated by one or more spaces. The parentheses can be placed in the same freeform argument PDB with the adjoining argument or can be in a separate freeform argument PDB.

For example, the argument string

```
¬STAC/CLOSED (GROS/CEO | GROS/PAY) ¬(PRIO/03 | PRIO/04)
```

can be entered in separate freeform argument PDBs like this:

```
¬STAC/CLOSED
(GROS/CEO
| GROS/PAY)
¬(PRIO/03
| PRIO/04)
```

or can be entered in separate freeform argument PDBs like this:

```
¬STAC/CLOSED
(
GROS/CEO
| GROS/PAY
)
¬(
PRIO/03
| PRIO/04
)
```

The argument can be entered in other ways as well, as long as the boolean operator (if one is present) appears first and no more than one argument is included in each freeform argument PDB.

You can also do text searching using the HLAPI. In order to do this, you must create a data view record for the type of record that you want to search for or else update an existing data view record to be used for the record inquiry and add the text index attribute record BLH&INDX. See the discussion on page 197 for a description of how you can do this.

Note: If you are using logical database partitioning (described in the *Tivoli Information Management for z/OS Program Administration Guide and Reference*), you should be aware that a HLAPI application cannot perform multipartition searches.

Each PDB on the input chain contains part of the search argument. The API does not validate freeform arguments.

You use the following HICA fields and PDBs for this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The following PDBs are required:

- TRANSACTION_ID must contain the 4-character transaction code HL11.
- Either PIDT_NAME or DATA_VIEW_NAME so that the HLAPI can perform data view processing. A static PIDT table or a data view record defines the view of the record the HLAPI processes. You can define just the fields that your application requires. See “Field Validation Using the Field Validation Module BLGPPFVM” on page 279 for additional information. If both PIDT_NAME and DATA_VIEW_NAME are specified, the HLAPI ignores PIDT_NAME.
  - PIDT_NAME must contain the alias or member name of the static retrieve PIDT table the HLAPI uses in processing the transaction. Member names are 1 to 7 uppercase characters long. Alias names are 32 uppercase characters long. You create static PIDTs by using the Table Build Utility.
  - DATA_VIEW_NAME contains a character field that specifies a data view name either as an alias or data view record ID. You must specify an alias name as a

32-character left-justified field exactly as it appears in a PALT. You specify a data view name as a 1- to 8-character name. If you use bypass panel processing, you must specify DATA_VIEW_NAME. A PIDT is generated from the data view record and associated data attribute and validation records.

Note: All PIDTs and related PIPTs can be maintained in storage to improve performance. This can be especially important if you are using data view records, as it can take a significant amount of time to generate a PIDT from the data model records.

- SEPARATOR_CHARACTER must contain the separator character value the HLAPI uses to process response data. If you omit this parameter, the HLAPI will end the transaction with an error.

The following PDBs are optional:

- APPLICATION_ID contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses. The application ID remains in effect until changed on a subsequent transaction. If APISECURITY=ON is specified in your BLX-SP startup parameters, you must ensure that the MVS user IDs running your application are allowed to use this application ID. “API Security” on page 287 contains additional information regarding API security.
- PRIVILEGE_CLASS contains a 1- to 8-byte uppercase startup privilege class name that the API passes to Tivoli Information Management for z/OS. This privilege class remains in effect until changed on a subsequent transaction. A privilege class name can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana. The application ID used must be an eligible user of this privilege class.
- ALIAS_TABLE must contain the 1- to 8- character uppercase name of the alias table used for this transaction. If you omit this parameter, or if it does not have a value, the HLAPI does not perform any alias table processing.
- ASSOCIATED_DATA must contain an uppercase identifier of the associated data item field that the HLAPI returns for each record found by the search. If you do not specify an alias table, field PDBDATA must contain an s-word or p-word index or the HLAPI does not return any data. If you do specify an alias table, field PDBDATA can contain an alias name, an s-word index, or a p-word index.

  Note: Associated data must be uppercase. The HLAPI attempts to retrieve this data item from all records found as a result of the inquiry and store its contents as part of the output PDB PDBDATA field.

  You cannot retrieve list item, phrase, and text data.
- EQUAL_SIGN_PROCESSING must contain the character value YES, which specifies that the HLAPI is to use equal sign processing. If you specify any other value, the HLAPI performs no equal sign processing.
- DATE_FORMAT must contain a supported date format. All dates passed between your application and the API will be in this format. Dates you pass into the API in this format will be converted to the primary date format of the database before being processed.

The following PDB is required when saving search results or when viewing previously saved search results:

- SEARCH_TYPE must contain a 1-byte character field indicating how the HLAPI treats this search. If this field is blank or set to S, it indicates to the HLAPI to start a new search. If this field is set to T, it indicates to the HLAPI to terminate an existing search. If this field is set to R, it indicates to the HLAPI to return matches from a saved search.
- NUMBER_OF_HITS must contain a 4-byte fixed field that specifies the maximum number of matches to be returned from a search. If this field contains a value, the actual number of matches is the smaller of:
  - The value in this field
  - The actual number of matches
  - the value in SORTPFX-N1.
- BEGINNING_HIT_NUMBER must contain a 4-byte fixed field that specifies the beginning match number to return. If your application specifies zero, the HLAPI uses a value of one.
- SEARCH_ID must contain a 4-byte fixed field containing the identifier of a search. It is assigned to either a new search results list or an existing search results list. If the value of this field is zero, the search results are not saved.

If you request previously saved search results, any new value that you specify for ASSOCIATED_DATA is ignored.

HICAINPP (INPUT)

The address of the first input PDB. The HLAPI uses PDBs specified on this chain to construct inquiry arguments. The PDB process option field PDBPROC determines whether the HLAPI adds the data parameter to the response buffer as a structured argument, as a freeform argument, or as a text search.

- The HLAPI processes structured (or quick search) arguments as follows:

  1. Locates the argument alias name specified in field PDBNAME in a given alias table (if using alias processing) or in the PIDT (if not using alias processing)
  2. Processes PIDT table entries for the argument.

  You cannot process Boolean or range operators when using this type of inquiry argument. If you specify such operators in the argument data, the HLAPI treats them as part of the argument data. You can use text item visible phrases as search arguments but you cannot use text data. Structured arguments can be validated by specifying V in PDB field PDBPROC. See PDBCODE on page 222 for code values returned by the API. Structured arguments are processed according to the setting of the **Cognize in mixed case?** option in the PIDT row or attribute record for the argument:

  - If **Cognize in mixed case?** is **Y**
    - If validation is requested, the case of the argument (after any adjustments made of the validation module based on the setting of the **Collected data case** option) will be used for the search.
    - If validation is not requested, the case of the argument as passed by the application will be used for the search. No case transformation will be done.
  - If **Cognize in mixed case?** is **N**

- Upper case will be used for the search, regardless of the case passed by the application and regardless of any adjustments made of the validation routine.

- The HLAPI processes freeform arguments (PDBPROC = F) by determining if a p-word alias name exists in field PDBNAME. The HLAPI uses the name in PDBNAME to locate its associated p-word stored in an alias table. If PDBNAME is not USE_AS_IS_ARGUMENT the HLAPI assumes that this is a p-word alias (an alias row containing a p-word). P-Words are alphanumeric phrases that end with a slash (/) or underscore (_) and that are not longer than 6 characters. The HLAPI stores the p-word in a PIAT entry row first, followed by the PDB argument data. If the HLAPI does not find a p-word, the HLAPI ends the transaction with an error. If field PDBNAME contains the reserved name USE_AS_IS_ARGUMENT, the HLAPI stores only the argument data in the LLAPI PIAT entry row.Freeform arguments are used as entered and must be provided by the application in the proper case.Freeform arguments can contain Boolean or range operators. When using these operators, the operator must be the first character of the data parameter. When the HLAPI builds the freeform argument, it stores the Boolean operator as the first character in the inquiry argument. This character is followed by the p-word data. The aggregate argument cannot contain imbedded blanks.

  Note: The aggregate length (Boolean, p-word, and data) cannot be greater than 33 characters, or the HLAPI ends the transaction with an error.

  The length of the field PIATDATA is 33 characters. The maximum number of characters available from a freeform argument segment for use in an inquiry is limited to the length of the key used to define the SDIDS. For example, if your application is searching a database with an SDIDS defined with a 32-byte key, a maximum of 32 bytes of each freeform argument segment is used to perform the inquiry.The HLAPI can combine structured and freeform arguments into a complete search argument. Structured arguments are always followed by the freeform arguments regardless of the order in which you specify them. Structured argument ordering is determined by the sequence in which they are defined in the PIDT rather than the sequence in which you specify them.

- The HLAPI processes text search arguments as described here. On the Record Inquiry transaction, you must specify the actual index name as an input PDB associated with S12E3. This PDB is required when TEXT_SEARCH_ARGUMENT is also supplied in an input PDB. The S12E3 input PDB contains a 1- to 8-character index record ID that Tivoli Information Management for z/OS uses to determine which Text Search index to search for the freeform text. These index names are defined:

  INDXSOLN
  Index for description and resolution freeform text for solution records.

  INDXPROB
  Index for description freeform text for problem records.

  INDXCHNG
  Index for description and resolution freeform text for change records.

  You can also have additional or alternate text search index names defined. On the HL11 transaction, text search arguments are specified in input PDBs. The field PDBNAME must contain the reserved name TEXT_SEARCH_ARGUMENT. These PDB fields should be set:

- The PDBNAME is TEXT_SEARCH_ARGUMENT.
- The PDBPROC value is X.
- The PDBDATW is the width of the text arguments This value cannot be larger than 132.
- The PDBDATL is the total length of the text arguments. PDBDATL must be an even multiple of PDBDATW.

You must use buffer processing with text arguments. The arguments can be sent in a single input PDB or in multiple input PDBs. Text arguments are used as entered. Text arguments can contain Boolean operators (AND, OR, AND NOT). Double quotation marks can be used to group text arguments together into search phrases, and parentheses can be used to group text arguments, search phrases, and Boolean operators together to form complex search arguments.

HICAOUTP (OUTPUT)

PDBs produced on this chain contain inquiry results data. When an inquiry generates results, the HLAPI creates an output PDB chain. The API names each output PDB INQUIRY_RESULT and each PDB contains data (describing one record in the search results list) extracted from an LLAPI PIRT row. The PDBAPPL field of the first output PDB contains a 4-byte fixed value defining the total number of matches for the search (from the LLAPI field PIRTSRRC). The HLAPI performs alias processing for each PIRT record type field (PIRTINDX) that it stores in a corresponding output PDB.

Use the following format for inquiry output field PDBDATA entries:

- 8-character external record identifier
- 32-character left-justified record type field alias name right-padded with blanks

  Note: If this field does not have an alias name, the HLAPI appends the record type s-word index for this field to the character S to provide an internal symbolic name that matches the format of a PIDT table symbolic name.
- 45-character left-justified associated data field right-padded with blanks.
- 2-character record processing code associated with a match entry. The API returns one of the following codes in this field:
  - 00 - No error was detected.
  - 01 - The record found 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 record.
  - 06 - Unknown problem when reading the record.

HICAMSGP (MESSAGES)

Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)

Contains the value previously stored by the HLAPI.

Table 62 on page 200, Table 63 on page 201, and Table 64 on page 202 show the Record Inquiry (HL11) transaction flows. In the tables, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL11. This method reduces the amount of text on the line in the tables. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields, and their parameters, see “HLAPI Structures” on page 216.

Table 62. HLAPI Transaction HL11. Record Inquiry for Viewing All Search Results

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDBs are required: – TRANSACTION_ID=HL11 – Either of the following: ■ PIDT_NAME = the member name or alias name of the PIDT the HLAPI uses to process the transaction ■ DATA_VIEW_NAME = the data view record ID or the alias of the data view record ID the HLAPI uses to process the transaction – SEPARATOR_CHARACTER = the character the HLAPI uses in processing response data The following PDBs are optional: – APPLICATION_ID = the application ID – PRIVILEGE_CLASS = the privilege class name – DATE_FORMAT = format to use for passing dates between your application and the API – ALIAS_TABLE = the name of the alias table used for this transaction. – ASSOCIATED_DATA = the identifier of a data item. When you do not specify an alias table, PDBDATA is a symbolic field index. When you do specify an alias table, PDBDATA is a symbolic field index or an alias name. – SEARCH_TYPE=blank or S – EQUAL_SIGN_PROCESSING = YES, which specifies that the HLAPI is to use equal sign processing. If you specify any other value, the HLAPI performs no equal sign processing. • HICAINPP = the address of the first input PDB ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Performs record inquiry ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP • HICAOUTP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. • HICAOUTP contains pointer to output PDB chain or 0000. ■ Continues processing.

Table 63. HLAPI Transaction HL11. Record Inquiry for Saving Search Results

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDBs are required: – TRANSACTION_ID=HL11 – Either of the following ■ PIDT_NAME = the member name or alias name of the PIDT the HLAPI uses to process the transaction ■ DATA_VIEW_NAME = the data view record ID or the alias of the data view record ID the HLAPI uses to process the transaction – SEPARATOR_CHARACTER = the character the HLAPI uses in processing response data The following PDBs are optional: – APPLICATION_ID = the application ID. – PRIVILEGE_CLASS = the privilege class name. – DATE_FORMAT = format to use for passing dates between your application and the API – ALIAS_TABLE = the name of the alias table used for this transaction. – SEARCH_ID=S to save a search – SEARCH_TYPE=S – BEGINNING_HIT_NUMBER = the beginning match number to return. If your application specifies 0, the HLAPI uses a value of 1. – NUMBER_OF_HITS = the maximum number of matches to be returned from a search. – EQUAL_SIGN_PROCESSING = YES, which specifies that the HLAPI is to use equal sign processing. If you specify any other value, the HLAPI performs no equal sign processing. See 146 for additional information on equal sign processing. • HICAINPP = the address of the first input PDB ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Performs record inquiry ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP • HICAOUTP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. • HICAOUTP contains pointer to output PDB chain or 0000. ■ Continues processing.

Table 64. HLAPI Transaction HL11. Record Inquiry for Viewing Saved Search Results

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDBs are required: – TRANSACTION_ID = HL11 – Either of the following ■ PIDT_NAME = the member name or alias name of the PIDT the HLAPI uses to process the transaction ■ DATA_VIEW_NAME = the data view record ID or the alias of the data view record ID the HLAPI uses to process the transaction – SEPARATOR_CHARACTER = the character the HLAPI uses in processing response data The following PDBs are optional: – PRIVILEGE_CLASS = the privilege class name. – ALIAS_TABLE = the name of the alias table used for this transaction. – SEARCH_ID = a search identifier required to return hits from an existing search (R) or end an existing search (T) – SEARCH_TYPE = R or T – BEGINNING_HIT_NUMBER = the beginning match number to return. If your application specifies 0, the HLAPI uses a value of 1. – NUMBER_OF_HITS = the maximum number of matches to be returned from a search. – EQUAL_SIGN_PROCESSING = YES, which specifies that the HLAPI is to use equal sign processing. If you specify any other value, the HLAPI performs no equal sign processing. • HICAINPP=the address of the first input PDB ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Performs record inquiry ■ Waits for completion ■ Sets following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP • HICAOUTP ■ Returns to application.
3	Application	■ Checks following fields set by server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. • HICAOUTP contains pointer to output PDB chain or 0000. ■ Continues processing.

Add Record Relation (HL12)

This transaction adds record relations to Tivoli Information Management for z/OS records. Use this transaction to create a relationship between a parent record and child records. For example, you can link a change record to change activity records. The transaction updates the parent record and adds nonreplaceable data items to the record.

Note: This is the only HLAPI transaction that adds nonreplaceable data to the database.

Each PDB identifies a type of relation data. For shipped parent record types, you can only add the names of the parent record's children or the identifiers of connected-to records. You can include multiple data items, delimited by a separator character, in the input PDB. For example, to add activities named ACT1 and ACT2 to the parent change record, you would set PDBDATA to 'ACT1,ACT2', assuming that the comma is the separator character.

You can direct the HLAPI to either retry this transaction from 1 to 255 times before returning control to your application or wait until the record is available. See page 18 for more information.

Checking out the record before the add record relation ensures that no other users can update the record prior to your update. Your administrator can define a time limit for checked out records (in the BSX-SP parameter APICHKOUTLIM described in the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*) so that records will not inadvertently remain indefinitely checked out if your application does not check in the record.

To create a parent and its child records:
1. Create the parent record (HL08).
2. Create each child record (HL08).
3. Add the relations data to the parent (HL12).

Note: If you are using logical database partitioning, you can perform an add record relation to a record only if the Owning Partition of that record matches the Primary Partition of your privilege class.

Use the following HICA fields and PDBs for this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The following PDBs are required:

- TRANSACTION_ID must contain the 4-character transaction code HL12.
- RNID_SYMBOL must contain a 1- to 8-character record identifier that is updated with relation data. A user-defined record identifier might have mixed data containing DBCS characters enclosed by an SO/SI pair.
- Either PIDT_NAME or DATA_VIEW_NAME so that the HLAPI can perform data view processing. A static PIDT table or a data view record defines the view of the record the HLAPI processes. You can define just the fields that your application requires. See "Field Validation Using the Field Validation Module BLGPPFVM" on page 279 for additional information. If both PIDT_NAME and DATA_VIEW_NAME are specified, the HLAPI ignores PIDT_NAME.
  - PIDT_NAME must contain the alias or member name of the static retrieve PIDT table the HLAPI uses in processing the transaction. Member names are 1 to 7 uppercase characters long. Alias names are 32 uppercase characters long. You create static PIDTs by using the Table Build Utility.
  - DATA_VIEW_NAME contains a character field that specifies a data view name either as an alias or data view record ID. You must specify an alias name as a 32-character left-justified field exactly as it appears in a PALT. You specify a data view name as a 1- to 8-character name. A PIDT is generated from the data

view record and associated data attribute and validation records. If you use bypass panel processing, you must specify DATA_VIEW_NAME.

Note: All PIDTs and related PIPTs can be maintained in storage to improve performance. This can be especially important if you are using data view records, as it can take a significant amount of time to generate the PIDT from the data model records.

- SEPARATOR_CHARACTER whose PDBDATA field contains the character field value the your application uses to separate each relations data item. If you omit this PDB, the HLAPI ends the transaction with an error.

The following PDBs are optional:

- APPLICATION_ID contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses. The application ID remains in effect until changed on a subsequent transaction. If APISECURITY=ON is specified in your BLX-SP startup parameters, you must ensure that the MVS user IDs running your application are allowed to use this application ID. “API Security” on page 287 contains additional information regarding API security.
- PRIVILEGE_CLASS contains a 1- to 8-byte uppercase startup privilege class name that the API passes to Tivoli Information Management for z/OS. This privilege class remains in effect until changed on a subsequent transaction. A privilege class name can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana. The application ID used must be an eligible user of this privilege class.
- A PDB named ALIAS_TABLE containing the uppercase name of the alias table used for this transaction. If you omit this parameter or if it does not have a value, the HLAPI does not perform any alias table processing. The field must be left-justified. See “Alias Tables” on page 238 for more information about alias processing.
- EQUAL_SIGN_PROCESSING must contain the character value YES, which specifies that the HLAPI is to use equal sign processing. If you specify any other value, the HLAPI performs no equal sign processing.

HICAINPP (INPUT)
Address of the first input PDB. PDBs found on this chain specify what data items are to be added to the record. These can be items such as child record names to store in the parent record. See PDBCODE on page 222 for code values returned by the API.

HICAOUTP (OUTPUT)
Contains the value previously stored by the HLAPI.

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 65 on page 205 shows the add record relation (HL12) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL12. This style reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is

the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields, and their parameters, see "HLAPI Structures" on page 216.

Table 65. HLAPI Transaction HL12. Add Record Relation

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDBs are required: – TRANSACTION_ID=HL12. – RNID_SYMBOL = the ID of the record receiving relations. – Either of the following: ■ PIDT_NAME = the name or alias of the static PIDT the HLAPI uses in processing the transaction. ■ DATA_VIEW_NAME = the data view record ID or the alias of the data view record ID the HLAPI uses in processing the transaction. If you use bypass panel processing, you must use DATA_VIEW_NAME. – SEPARATOR_CHARACTER = the character used by the HLAPI in processing record relations data. The following PDBs are optional: – APPLICATION_ID = the application ID – PRIVILEGE_CLASS = the privilege class name – ALIAS_TABLE = the name of the alias table used in processing this transaction – EQUAL_SIGN_PROCESSING = YES, which specifies that the HLAPI is to use equal sign processing. If you specify any other value, the HLAPI performs no equal sign processing. • HICAINPP = address of the first input PDB ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Adds record relations ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

Delete Record (HL13)

This transaction deletes a Tivoli Information Management for z/OS record.

Note: If you are using logical database partitioning, you can delete a record only if the Owning Partition of that record matches the Primary Partition of your privilege class.

Use the following HICA fields and PDBs for this transaction:

HICAENVP
Must contain the value stored on completion of HLAPI initialization.

HICACTLP (CONTROL)
The following PDBs are required:

- TRANSACTION_ID must contain the 4-character transaction code HL13.
- RNID_SYMBOL must contain the 1- to 8-character external record identifier of the record to be deleted. A user-defined record identifier can have mixed data containing DBCS characters enclosed by an SO/SI pair.

The following PDBs are optional:

- APPLICATION_ID contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses. The application ID remains in effect until changed on a subsequent transaction. If APISECURITY=ON is specified in your BLX-SP startup parameters, you must ensure that the MVS user IDs running your application are allowed to use this application ID. “API Security” on page 287 contains additional information regarding API security.
- PRIVILEGE_CLASS contains a 1- to 8-byte uppercase startup privilege class name that the API passes to Tivoli Information Management for z/OS. This privilege class remains in effect until changed on a subsequent transaction. A privilege class name can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana. The application ID used must be an eligible user of this privilege class.

HICAINPP (INPUT)
Initialize to zeros.

HICAOUTP (OUTPUT)
Contains the value previously stored by the HLAPI.

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 66 shows the delete record (HL13) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL13. This method reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields, and their parameters, see “HLAPI Structures” on page 216.

Table 66. HLAPI Transaction HL13. Delete Record

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDBs are required: – TRANSACTION_ID=HL13 – RNID_SYMBOL = the ID of the record to be deleted The following PDBs are optional: – APPLICATION_ID = the application ID – PRIVILEGE_CLASS = the privilege class name ■ BLGYHLPI(*HICA*).

Table 66. HLAPI Transaction HL13 (continued). Delete Record

Step	Program	Action
2	Server	■ Validates HICA and PDB fields ■ Deletes record ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAMSGP • HICAERRP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

Get Data Model (HL31)

This transaction returns selected information from the specified static PIDT or generated PIDT. It is used by a HLAPI application program to obtain information on the fields in a particular data view record or PIDT. To invoke this transaction, an application program specifies the static PIDT name or data view name from which to retrieve information, and optionally, an alias table to convert PIDTSYMB values into more meaningful field names. On return from the HL31 transaction, the application program loops through the output PDBs looking for the PDB name that corresponds to the field sought by the application. The application then parses the data in the PDBDATA field using the external BLGUHIDM data mapping.

If the application has PIPT validation data returned, the application obtains the next PDB to see if the PDBNAME field contains the same PIDTSYMB value and a value of V in the PDBTYPE field. If so, the application parses the data in the PDBDATA field using the HIVP data mapping. The application continues to retrieve the validation patterns until the PDBNAME field of the next PDB contains a different PIDTSYMB name. The validation patterns retrieved can be used by the application to validate the field data before the data is sent to the server.

The output that results from this transaction is one output PDB for each static PIDT row (if you specify PIDT_NAME) or one output PDB for each data attribute record (if you specify DATA_VIEW_NAME); each PDB so produced is followed by one output PDB per PIPT row associated with the PIDT symbol name (s-word). The PDBNAME field of each output PDB is the PIDTSYMB value (the symbolic s-word or p-word) or the alias name for the PIDTSYMB value (if ALIAS_TABLE is specified and there is an alias name defined for the field). The PDBTYPE field is set to F for PIDT data and is set to V for PIPT validation data.

For PIDT output PDBs, the PDBDATA field is mapped by the HIDM. The fields in the HIDM, described in Table 70 on page 237, are a subset of fields from the PIDT structure. PIDTMNCR and PIDTMAXL are converted to a character and are stored into HIDMNCR and HIDMAXL, respectively.

Note: The value for HIDMREQD is always returned as N.

For PIPT output PDBs, the PDBDATA field is mapped by the HIVP, described in Table 71 on page 238. The fields in the HIVP are a subset of fields from the PIPT structure. PIPTNUMR is converted to character and stored into HIVPNUMR.

Use the following PDBs for this transaction:

HICACTLP (CONTROL)
The following PDBs are required:

- TRANSACTION_ID must contain the 4-character transaction code HL31.
- You must specify where the data model information is to come from by specifying either a PIDT_NAME to define the static PIDT or a DATA_VIEW_NAME to specify the data view (if both are specified, the DATA_VIEW_NAME is used).

The following PDBs are optional:

- APPLICATION_ID contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses. If APISECURITY=ON is specified in your BLX-SP startup parameters, you must ensure that the MVS user IDs running your application are allowed to use this application ID. "API Security" on page 287 contains additional information regarding API security.
- PRIVILEGE_CLASS contains a 1- to 8-byte uppercase privilege class name that the API passes to Tivoli Information Management for z/OS. This privilege class remains in effect until changed on a subsequent transaction. A privilege class name can be mixed data containing DBCS characters enclosed by an SO/SI pair, but it cannot contain SBCS Katakana. The application ID used must be an eligible user of this privilege class.
- ALIAS_TABLE is used to resolve PIDT names and field names. If specified, output PDBs will be named using the alias name for a field, if one exists, or the PIDTSYMB value for the fields.
- RETURN_VALIDATION_DATA contains the character value YES or NO, which indicates whether to return validation pattern data. The default value is YES.

HICAINPP (INPUT)
Initialize to zeros.

HICAOUTP (OUTPUT)
Output PDBs are returned that describe the data model.

HICAMSGP (MESSAGES)
Contains the value previously stored by the HLAPI.

HICAERRP (ERROR CODES)
Contains the value previously stored by the HLAPI.

Table 67 on page 209 shows the get data model (HL31) transaction flow. In the table, symbolically named PDBs have a value, for example, PDB TRANSACTION_ID=HL31. This method reduces the amount of text on the line in the table. The name on the left of the equation is the value in the PDB field PDBNAME. The data on the right of the equation is the value in the PDB field PDBDATA. For more detailed information on the HLAPI structures, their fields, and their parameters, see "HLAPI Structures" on page 216.

Table 67. HLAPI Transaction HL31. Get Data Model

Step	Program	Action
1	Application	■ Sets fields as follows: • HICACTLP (pointer to first control PDB) The following PDBs are required: – TRANSACTION_ID=HL31 – PIDT_NAME of the static PIDT or DATA_VIEW_NAME of the data view The following PDBs are optional: – APPLICATION_ID = the application ID – PRIVILEGE_CLASS = the privilege class name – ALIAS_TABLE = the name of an alias table – RETURN_VALIDATION_DATA = YES or NO to specify whether to return validation pattern data ■ BLGYHLPI(*HICA*).
2	Server	■ Validates HICA and PDB fields ■ Gets data model information ■ Waits for completion ■ Sets the following HICA fields: • HICARETC • HICAREAS • HICAOUTP • HICAMSGP • HICAERRP ■ Returns to application.
3	Application	■ Checks the following fields set by the server: • HICARETC contains return code. • HICAREAS contains reason code. • HICAOUTP contains pointer to output PDB chain or 0000. • HICAMSGP contains pointer to message PDB chain or 0000. • HICAERRP contains pointer to error PDB chain or 0000. ■ Continues processing.

HLAPI Graphic Examples

The following figures show graphic representations of parameter data that the HLAPI uses for some of the transactions described previously in the chapter. Each figure shows representative PDBs on each PDB chain.

It is assumed that your application creates control and input chains before starting the transaction. The HLAPI creates output, message, and error chains while the transaction runs. Each of the following examples shows a view of these chains after a transaction completes.

Initialize Tivoli Information Management for z/OS

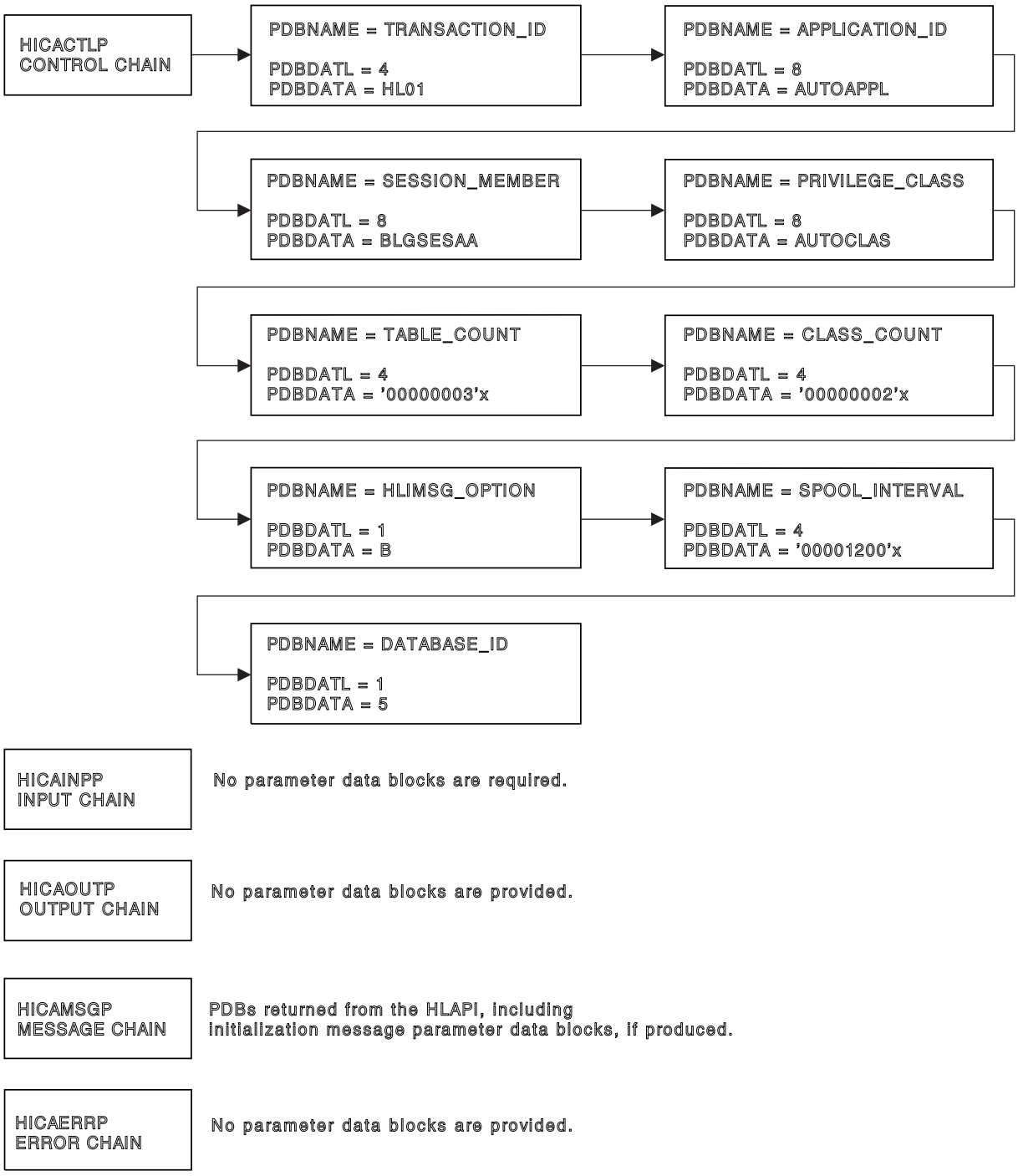

Figure 7. Initialize Tivoli Information Management for z/OS Example

Record Retrieve

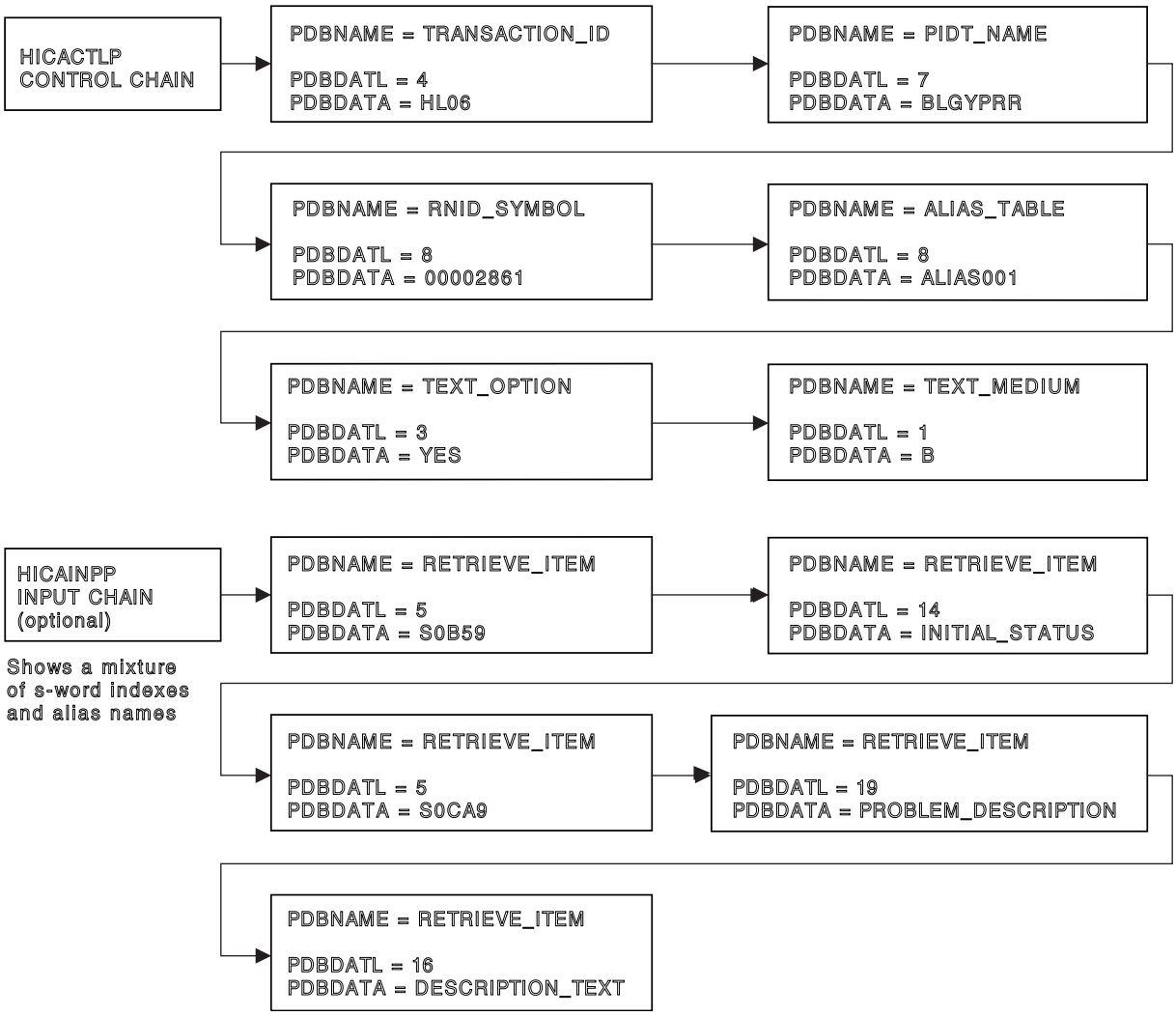

Figure 8. Retrieve Record Example (Part 1 of 2)

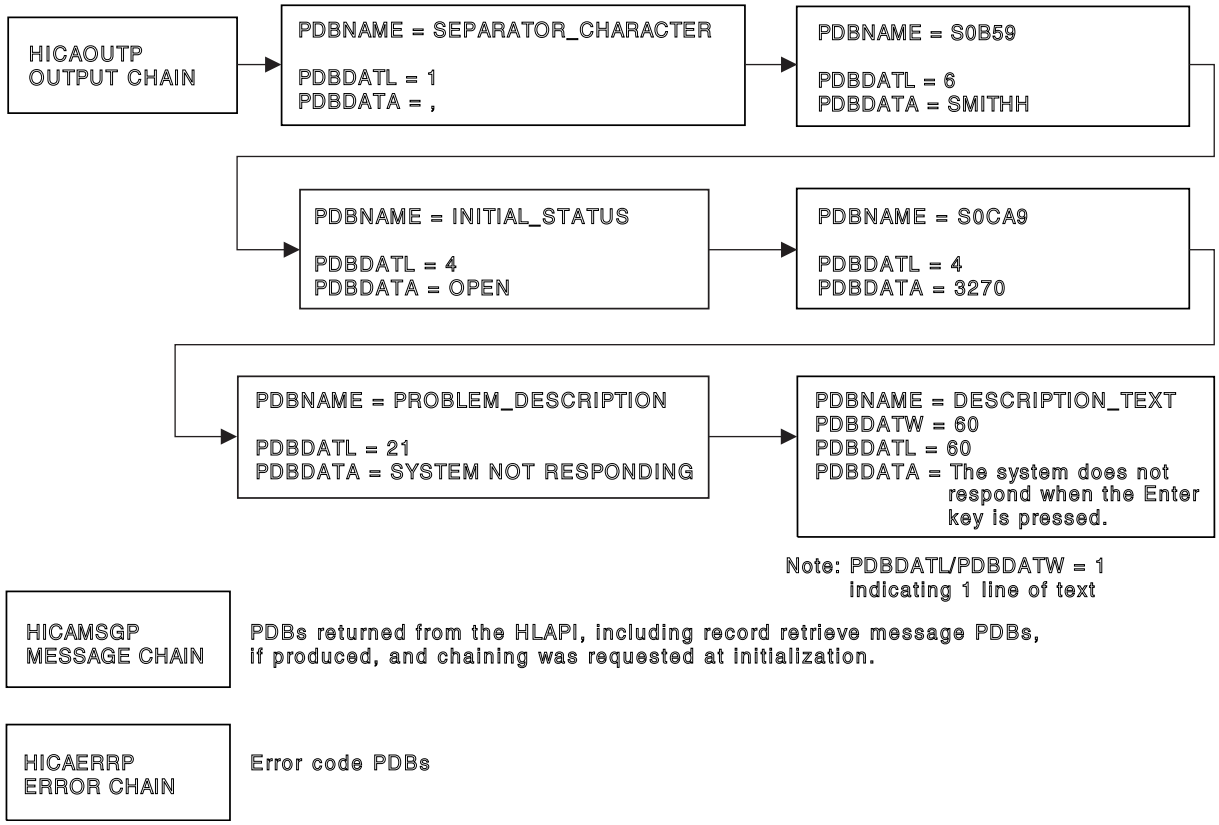

Figure 8. Retrieve Record Example (Part 2 of 2)

Create Record

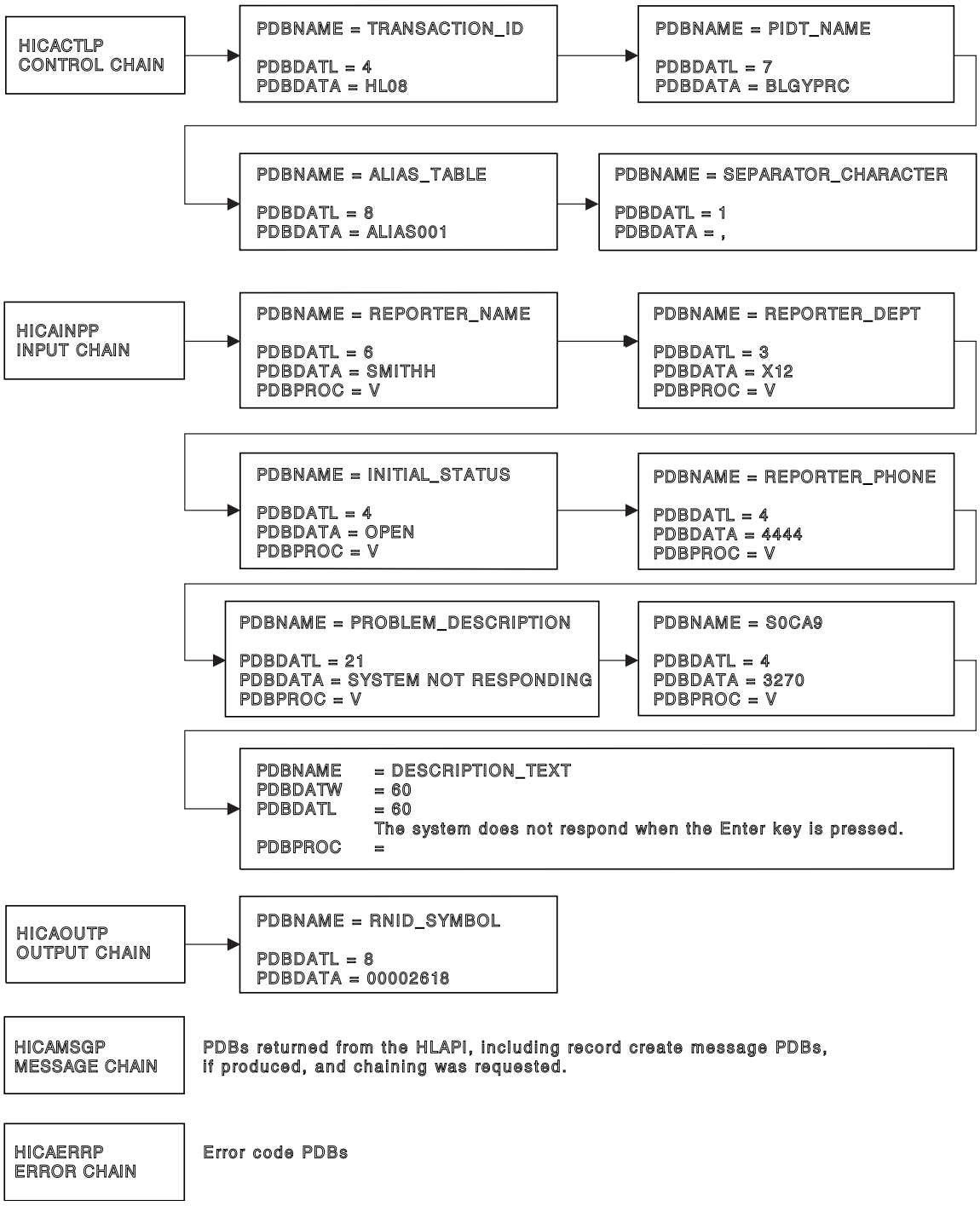

Figure 9. Create Record Example

Record Inquiry

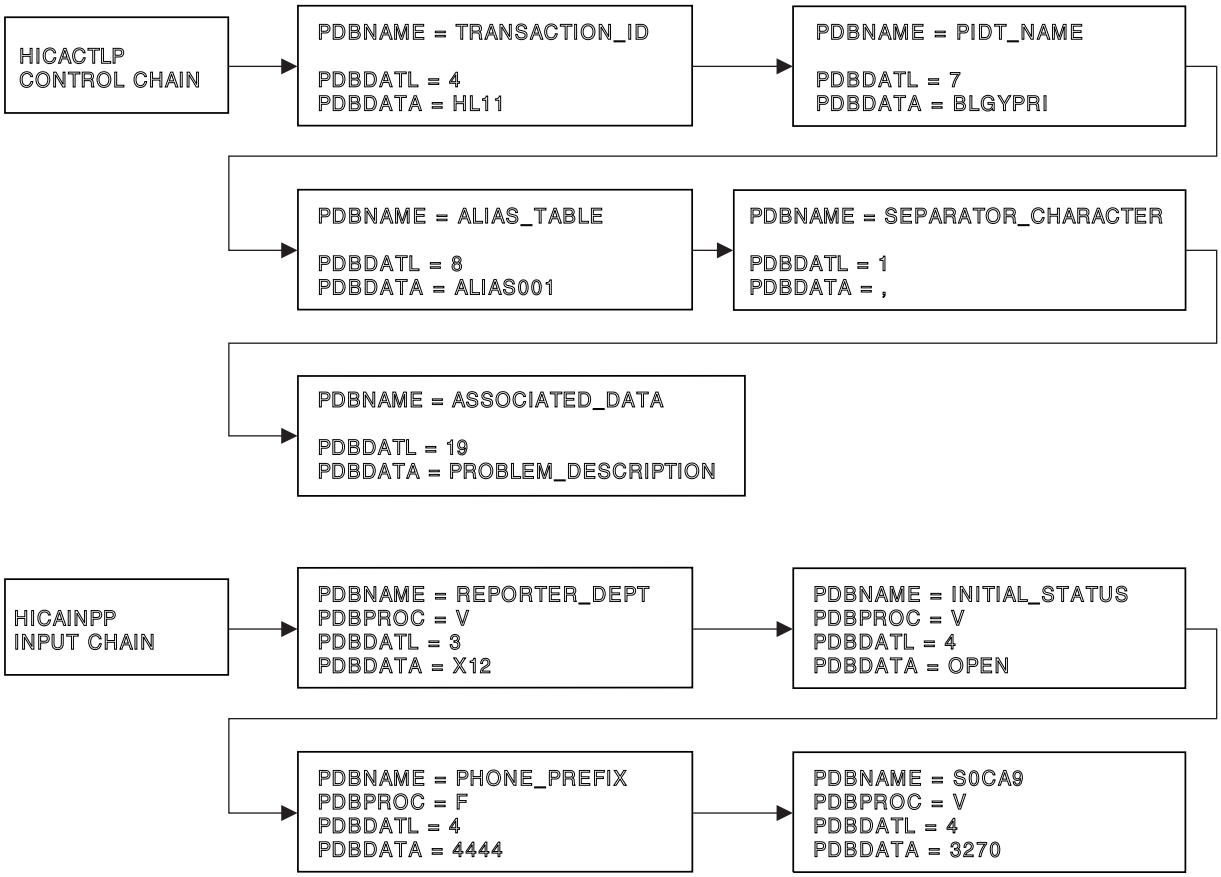

Figure 10. Record Inquiry Example (Part 1 of 2)

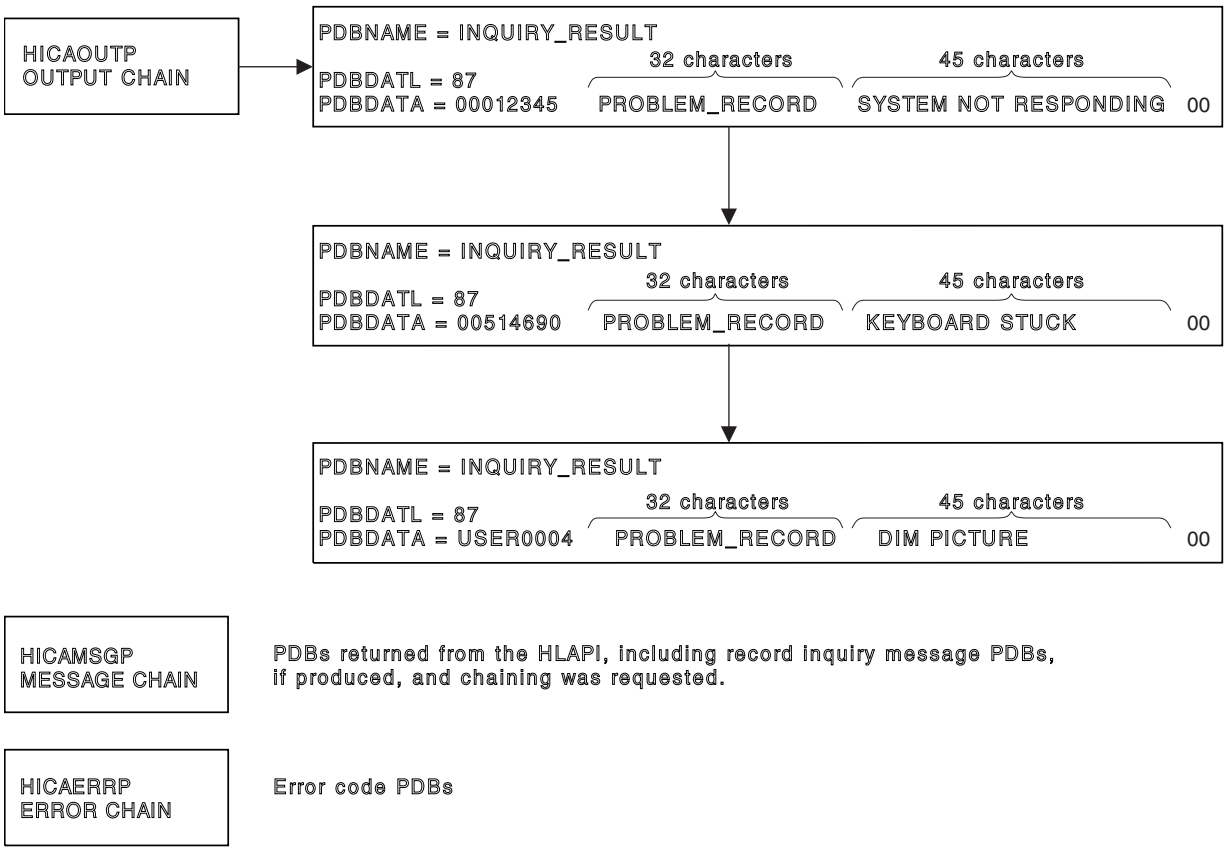

Figure 10. Record Inquiry Example (Part 2 of 2)

Delete Text Data Set

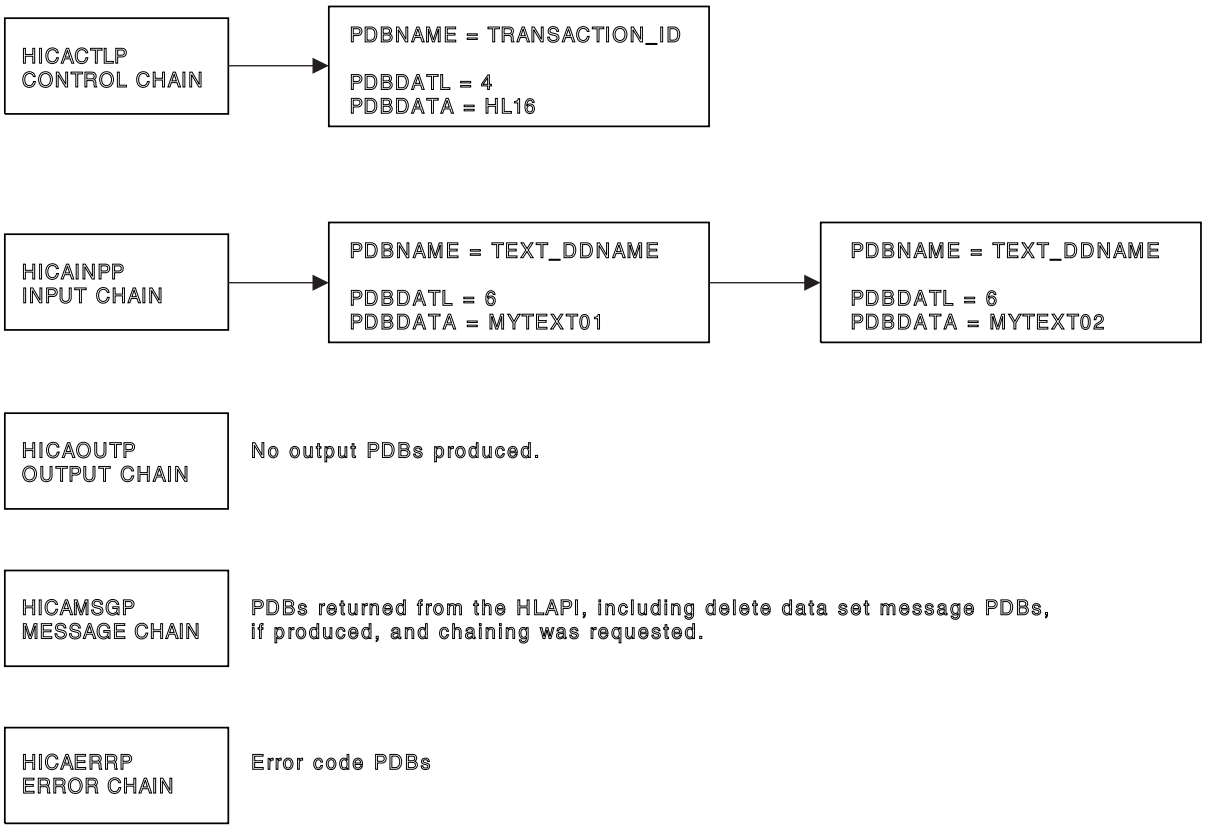

Figure 11. Delete Text Data Set Example

HLAPI Structures

Your application uses two HLAPI structures to access the Tivoli Information Management for z/OS database through the HLAPI. These structures are:

- The HLAPI communications area (HICA)
- The parameter data block (PDB).

High-Level Application Program Interface Communications Area

Your application allocates the HICA. It is used to communicate between the HLAPI and your application. The HICA also serves as a communications anchor to the HLAPI PDB structures. You can find a sample assembler DSECT mapping for the HICA in the sample library (SBLMMACS) that is part of Tivoli Information Management for z/OS. Look for BLGUHICA in this sample library.

Table 68 on page 217 shows the structure of the HICA and the page number where the table fields are explained:

Note: As shown in the table, some fields are set by the interface. Do not design your application to set these fields; if it does, unpredictable results occur.

Table 68. HICA Field Definitions

Field Label	Offset DEC(HEX)	Length DEC	Description	Set by	page
HICAACRO	0(0)	4	The acronym HICA	Application	217
HICALENG	4(4)	4	Length of this structure (fixed)	Application	217
HICARETC	8(8)	4	Transaction return code (fixed)	Interface	217
HICAREAS	12(C)	4	Transaction reason code (fixed)	Interface	217
HICAENVP	16(10)	4	Transaction environment anchor (pointer)	Interface	217
HICACTLP	20(14)	4	Control PDB anchor (pointer)	Application	218
HICAINPP	24(18)	4	Input PDB anchor (pointer)	Application	218
HICAOUTP	28(1C)	4	Output PDB anchor (pointer)	Interface	218
HICAMSGP	32(20)	4	Message PDB anchor (pointer)	Interface	218
HICAERRP	36(24)	4	Error PDB anchor (pointer)	Interface	218
HICASTPA	40(28)	4	Subtask TCB address place holder address (pointer)	Interface	218
HICACRRC	44(2C)	4	Reserved for HLAPI use.	Interface	218
HICARESV	48(30)	32	Server reserved. Must be zeros.	Application	218

The following section describes the purpose of each field in the HICA structure:

HICAACRO
A 4-character field containing the character string HICA to identify this communication area. Your application sets this field.

HICALENG
A 4-byte fixed field that contains the length of this structure. This value, along with the value in HICAACRO, is used to validate the structure when your application passes it to the HLAPI. Your application sets this field.

HICARETC
A 4-byte fixed field containing a return code from the HLAPI. The API sets this field. See “Return and Reason Codes” on page 301 for a list of return codes.

HICAREAS
A 4-byte fixed field containing the reason code from the HLAPI. The API sets this field. See “Return and Reason Codes” on page 301 for a list of reason codes.

HICAENVP
A 4-byte pointer field containing the address of the HLAPI environment area.

Note: Initialize this field to zero when your application passes it to the HLAPI for the first time during an initialize Tivoli Information Management for z/OS transaction (HL01). During the initialize Tivoli Information Management for z/OS transaction, the HLAPI sets this field.

Your application must maintain the address stored in this pointer until the Terminate Tivoli Information Management for z/OS (HL02) transaction is complete.

HICACTLP – first CONTROL PDB
A 4-byte pointer containing the address of the first control PDB the HLAPI processes. Your application sets this field.

HICAINPP – first INPUT DATA PDB
A 4-byte pointer containing the address of the first input PDB the HLAPI processes. Your application sets this field.

HICAOUTP – first OUTPUT DATA PDB
A 4-byte pointer containing the address of the first output data PDB the HLAPI creates. Output PDB chains are freed at the time of the next HLAPI call. The API sets this field.

HICAMSGP – first MESSAGE DATA PDB
A 4-byte pointer containing the address of the first message data PDB the HLAPI creates. Message PDB chains are freed at the time of the next HLAPI call. The API sets this field.

HICAERRP – first ERROR CODE PDB
A 4-byte pointer containing the address of the first ERROR CODE PDB the HLAPI creates. Error PDB chains are freed at the time of the next HLAPI call. The HLAPI sets this field.

HICASTPA
A 4-byte pointer containing the address of the storage containing the address of the subtask TCB.

Note: If your application uses an ESTAE exit, detach the Tivoli Information Management for z/OS subtask if appropriate as follows:
1. Check that there is a subtask TCB address.
2. Issue a DETACH.

The field pointed to contains either the address of the subtask TCB (if the subtask is active) or zero (if the subtask is inactive). The API sets this field.

HICACRRC
Reserved for server use.

HICARESV
A 32-byte area reserved for future use. Your application must initialize this area to zeros.

Parameter Data Block

The PDB structure passes data between the HLAPI and your application. Your application must allocate and initialize a PDB for each item of data that your application passes to the HLAPI. The HLAPI allocates and initializes a PDB for each item of data that it passes back to your application. If the API determines that the data structures passed to it are valid, it frees the output, message, and error PDB chains from the previous transaction.

Your application can pass two chains of PDBs to the HLAPI.

- The first chain contains control parameters that the HLAPI uses to determine which transaction to process and how to process it.
- The second chain contains input (models of data) record data. Create, update, add, and inquiry transactions require input record data.

The HLAPI can pass three chains of PDBs to the application. The first chain specifies output data for output-generating transactions, such as the Retrieve Record (HL06) transaction. The second chain specifies message data associated with running the transaction. The third chain specifies error data associated with running the transaction.

The HLAPI initializes HICAOUTP, HICAMSGP, HICAERRP, HICARETC, and HICAREAS to 00 when it receives control at the start of a transaction.

Table 69 shows the structure of the parameter data block and the page number where the data block fields are explained:

Table 69. Parameter Data Block Field Definitions

Field Label	Offset DEC(HEX)	Length DEC	Description	page
PDBNEXT	0(0)	4	Address of next PDB in the chain (pointer)	219
PDBPREV	4(4)	4	Optional address of previous PDB in the chain (pointer)	219
PDBACRO	8(8)	4	Parameter Data Block Acronym (character string of PDB left-justified and right-padded with a blank)	219
PDBNAME	12(C)	32	Parameter data symbolic name (character)	220
PDBTYPE	44(2C)	1	Parameter data type (character)	220
PDBPROC	45(2D)	1	Parameter data processing flag (character)	221
PDBCODE	46(2E)	1	Parameter data error code (character) (initialized to blank by application)	222
PDBRSV1	47(2F)	5	Reserved	223
PDBDATW	52(34)	4	Parameter data unit width (fixed)	223
PDBAPPL	56(38)	4	For use by the creator of the PDB	223
PDBDATL	60(3C)	4	Length of parameter data (fixed)	223
PDBDATA	64(40)	variable	Parameter data (character and fixed)	223

The following section describes the purpose of each PDB field:

PDBNEXT

A 4-byte pointer to the address of the next PDB on this chain. This required field contains zeros if there are no additional PDBs on the chain.

Note: Do not alter this field for any PDB chains produced by the HLAPI.

PDBPREV

A 4-byte pointer to the address of the previous PDB on this chain. This optional backward pointer field contains zeros if the chain contains no previous PDBs.

Note: Do not alter this field for any PDB chains produced by the HLAPI.

PDBACRO

A 4-byte character field containing the string PDB left justified and right padded with a blank. It identifies this control structure. This identifier value must appear in all PDB structures used by the HLAPI.

PDBNAME

A 32-byte character field containing the symbolic name of the parameter data item this PDB contains. Symbolic names are left-justified uppercase character strings padded with blanks. These strings can only contain the characters A-Z, 0-9, an underscore, and a period. The first character must be an alphabetic character (A-Z). Symbolic names cannot contain imbedded blanks or DBCS characters enclosed by an SO/SI pair.

This required field contains one of the following:

- S-Word index (used for input, output, and error PDBs)
- P-Word index (used for input PDBs)
- Alias external name (used for input and output PDBs)
- Reserved PDB external name (used for control, input, and message PDBs).
- History data retrieved and returned as output data is HISTORYNNNNNN where NNNNNN is initialized at 000001 and is incremented by 1 for each history data item.

The name you specify in a PDB depends on its use. Control and message PDBs always use reserved names. See "Reserved Symbolic PDB Names" on page 224 for a list of reserved names. Input and output PDBs can use various symbolic names depending on the transaction specified.

When a PDB refers to a freeform inquiry argument, this field must contain either the alias name of the Tivoli Information Management for z/OS p-word used to construct the argument or the reserved symbolic name USE_AS_IS_ARGUMENT.

PDBTYPE

A 1-byte character field indicating the type of data that this PDB contains. The HLAPI uses this field for input and output PDB processing. The following data type field values are accepted:

- Blank – No data type assigned to this parameter

  The HLAPI returns this value to output PDBs that relate to data items that are single word response fields such as reporter name or status. The HLAPI also returns this value to message and error PDBs.

- A – Direct-add response data

  The HLAPI returns this value to an output PDB when the HLAPI determines that PDBDATA contains a direct-add item.

- D – Date field response data

  The HLAPI returns this value to an output PDB when the HLAPI determines that PDBDATA field contains a date. Date response fields contain date data in a format unique to your company's needs. This code alerts your application to possibly process the date data in a unique way.

- F – Data Model information. Used when the Get Data Model transaction is returning information from BLGPIDT. The PDBDATA field is mapped by BLGUHIDM.

- G – Specifies that this PDB comes first in a group of one or more related history data items. This is indicated by the associated PIHTSGRP row field set to Y.

- H – Specifies that this PDB is not the first PDB in a group of several related history data items. This is indicated by the associated PIHTSGRP row field not set to Y.
- L – List item field response data

  The HLAPI returns this value to an output PDB when the HLAPI determines that PDBDATA contains list item responses. A list item is a field possibly containing multiple responses produced by the list processor. The list processor displays data in tabular form. The purpose of this type of value is to alert your application that the data field might contain multiple responses separated by the separator character defined in the output PDB named SEPARATOR_CHARACTER.
- M – When data is returned to an application by using the SETAPIDATA control line in a HLAPI, data is returned in the form of output PDBs. The SETAPIDATA can build output PDBs that contain a single string as output data or multiple lines of output data. PDBTYPE is set to M if the PDB contains multiple lines of data.
- P – Phrase item

  The HLAPI returns this value to an output PDB when the HLAPI determines that PDBDATA contains a keyword phrase or visible phrase item. *Visible phrases* are external descriptions. *Keyword phrases* are associated with selections made during the interactive collection of data. This type of data is useful in determining if a certain panel dialog was entered. An example of a visible phrase is 'Reporter data' from the problem summary panel. An example of a keyword phrase is RECS=PROBLEM collected when selecting a problem record as the type of record you want to create.
- S – String field response data

  The HLAPI returns this value to an output PDB when the HLAPI determines that the field was defined as a string in the PIDT or data view. The problem 'Description' field is an example of a string field.
- V – Validation pattern information. Used when the Get Data Model transaction is requested to return validation pattern information from BLGPIPT. The PDBDATA field is mapped by BLGUHIVP.
- X – Text related data

  The HLAPI returns this value to an output PDB when the HLAPI determines that PDBDATA contains text related data. Field PDBDATA contains either data set name information or text, depending on which text processing options you specify.

PDBPROC

A 1-byte character field indicating the type of processing applied to the data that this PDB refers to when the PDB is used as input to the HLAPI.

Valid field values are:

- Blank – No unique processing on this data.
- V – Perform data response validation and case conversion when processing this PDB.

Note: The HLAPI does not validate string, phrase, text, and direct-add items, but it does perform case conversion of string data.

- F – Process as a freeform inquiry argument. If you use this value for anything other than inquiry transactions (HL11), it is ignored and not treated as an error.

  The HLAPI converts freeform inquiry arguments to LLAPI PIAT entries. When Tivoli Information Management for z/OS processes freeform arguments using an alias name in PDBNAME for a p-word, the alias table row is located in the alias table. The HLAPI extracts the corresponding p-word from the table and stores it first in an LLAPI PIAT entry followed by the argument data contained in PDBDATA. This gives you a way to symbolically name the p-word appended in front of the argument data. If PDBNAME contains the reserved name USE_AS_IS_ARGUMENT, then only the argument data is stored in the LLAPI PIAT entry. The argument data can contain a p-word, for example, PERS/JON, or just be data.

  Freeform arguments can also contain Boolean or range operators. When using these operators, the operator must be the first character of the argument data and the data must not contain any imbedded blanks. When the HLAPI builds the freeform argument, it stores the Boolean operator as the first character in the LLAPI PIAT entry followed by the p-word and then the data.

  Note: You cannot use the F processing flag for phrase or direct-add items.

- T – Perform PDB data logging. The HLAPI uses this field in the TRANSACTION_ID PDB with the initialize Tivoli Information Management for z/OS transaction (HL01) to determine what additional data to log. See the initialize Tivoli Information Management for z/OS transaction (HL01) on page 153 for more information. This value is ignored for any transaction other than initialize Tivoli Information Management for z/OS transaction (HL01).

- X – Perform Text searching using HL11. See the record inquiry transaction (HL11) on page 194 for more information.

PDBCODE

A 1-byte character field set by the HLAPI in an input PDB. Your application must set this field to blank so that the HLAPI can set this field when one of the following errors occurs:

- Blank – No error found.
- E – There is no response data for this item in the record. This is not an error; it is only an indication that the field is null.
- I – the response data parameter found internal processing errors or a text item found data set processing errors. This can also mean that a text data item was requested using a RETRIEVE_ITEM PDB with TEXT_MEDIUM set to D.
- L – The response data parameter is too long or is larger than the size of the data that can be collected by Tivoli Information Management for z/OS. This code indicates that the response data is greater than the size of the PIDT field or the PIAT entry argument is greater than 33 bytes.
- M – the response data item could not be found in the currently specified PIDT or PALT. The p-word could not be found in the PALT. This could also occur if you specify alias names without having specified alias processing.

- N – The number of responses for a field is larger than the maximum defined for the field in PIDTMNCR.
- V – The response data parameter did not meet validation specifications or was used incorrectly with a transaction:
  - The data is not a valid date
  - The type of the data returned did not match the PIDT definition of the data.
  - PIDTCURL divided by PIDTCNFR produced a nonzero remainder when processing text units.

PDBRSV1
A 5-byte area reserved for future use.

PDBDATW
A 4-byte fixed field specifying the width of a data field unit that this PDB references. Only text data fits this category. If control PDB TEXT_STREAM is not YES, PDBDATW is used as follows. When this PDB references text data, this field specifies the width of a text unit, and PDBDATL specifies the total length of all the text units. This PDB is also used when your application inputs blocks of text to define the width of a text line in the block. If control PDB TEXT_STREAM is YES, both PDBDATW and PDBDATL equal the total length of the text. When this PDB contains text data set name information, this field is always zero. The fields PDBDATW and PDBDATL are important for applications retrieving records and disassembling the text.

PDBAPPL
A 4-byte field that the creator of this PDB, whether your application or the HLAPI, can use for any purpose. When performing a search, the HLAPI uses this field on the first output PDB to return the total number of matches from that search.

PDBDATL
A 4-byte fixed field containing the length of the data that this structure references.

If this API connects to a BLX-SP that supports DBCS (that is, DBCS=YES is specified in the BLX-SP parameters), the maximum value of this field is 32 767.

Note: Your application can specify zero in this field for control PDBs to ignore the PDB. This method is useful when processing control PDBs because your application can create an initial control PDB chain once and change values as it processes.

PDBDATA
A variable length character or fixed field containing the data that this structure references. Only control data can be fixed. Input response and control data must be uppercase to be consistent with interactive response processing.

This is the only field that can contain mixed data with DBCS characters enclosed by an SO/SI pair. Examples of mixed data fields are external record identifier (user defined) and privilege class name.

If this API connects to a BLX-SP that supports DBCS (that is, DBCS=YES is specified in the BLX-SP parameters), the maximum length of the data pointed to by this field is 32 767.

PDB Example

This is a sample view of a PDB with pertinent fields set.

```
PDBNEXT = 00031000 (address of next PDB)
PDBPREV = 00000000 (address of previous PDB)
PDBACRO = PDB
PDBNAME = REPORTER_NAME
PDBTYPE = blank
PDBPROC = V (validate the response)
PDBCODE = blank
PDBRSV1 = 0000000000
PDBDATW = 00000000
PDBDAPPL= 00000000
PDBDATL = 00000006
PDBDATA = BROWNJ
```

Reserved Symbolic PDB Names

The following PDB symbolic names are reserved for use by the HLAPI. They are intended to be used for control PDBs or where the HLAPI must process PDBs that use unique names. You can use these names in input PDBs, but do not use them where the interface expects to process a unique name; for example, USE_AS_IS_ARGUMENT for inquiry input processing. If you use these names in this way, unpredictable results can occur.

- ALIAS_TABLE
- APIMSG_OPTION
- APPLICATION_ID
- APPROVAL_STATUS
- ASSOCIATED_DATA
- BEGINNING_HIT_NUMBER
- BYPASS_PANEL_PROCESSING
- CICS_CM_TIME_OUT_VALUE
- CICS_INTER_TIME_OUT_VALUE
- CICS_PARTNER_ID
- CICS_USER_ID
- CLASS_COUNT
- DATA_VIEW_NAME
- DATABASE_ID
- DATABASE_PROFILE
- DATE_FORMAT
- DEFAULT_DATA_STORAGE_SIZE
- DEFAULT_OPTION
- DELETE_HISTORY
- EQUAL_SIGN_PROCESSING
- HIGH_MEMORY
- HISTORY_DATA
- HLAPILOG_ID
- HLIMSG_OPTION
- INQUIRY_RESULT
- LIST_MODE
- MESSAGE_DATA
- MULTIPLE_RESPONSE_FORMAT
- NUMBER_OF_HITS
- PASSWORD
- PIDT_NAME
- PRIVILEGE_CLASS
- REPLACE_TEXT_DATA
- RETRIEVE_ITEM
- RETURN_VALIDATION_DATA

- RNID_SYMBOL
- SEARCH_ID
- SEARCH_TYPE
- SECURITY_ID
- SEPARATOR_CHARACTER
- SESSION_MEMBER
- SPOOL_INTERVAL
- TABLE_COUNT
- TEXT_AREA
- TEXT_AUDIT_OPTION
- TEXT_DDNAME
- TEXT_MEDIUM
- TEXT_OPTION
- TEXT_STREAM
- TEXT_SEARCH_ARGUMENT
- TEXT_UNITS
- TEXT_WIDTH
- TIMEOUT_INTERVAL
- TIME_ZONE
- TRANSACTION_ID
- TSP_NAME
- USE_AS_IS_ARGUMENT
- USER_PARAMETER
- USER_PARAMETER_DATA

Parameter Data Definition

The following section describes parameter data passed within the control, input, output, message, and error PDB chains used by the HLAPI.

Note: Control and message PDBs always use reserved symbolic names.

CONTROL chain

Control PDB parameters tell the HLAPI which transaction to process and how to process it. The HICA field HICACTLP points to the first control PDB on the chain of control PDBs. Parameter data specified on the control PDB chain is unique to the transaction being performed. Specify character data in uppercase with no imbedded blanks only.

Note: Parameter data must be left justified in the PDBDATA field and right padded with blanks.

You can specify the following parameters on the control PDB chain. See Table 49 on page 151 to find where to go in this chapter to determine which control parameters the HLAPI uses for each transaction. The listed reserved names are stored in the PDB field PDBNAME.

ALIAS_TABLE

Contains a 1- to 8-character alias table name used in processing the transaction. If your application does not specify this parameter, the HLAPI does not perform alias table processing for the transaction. Use this parameter with TABLE_COUNT. If your application omits TABLE_COUNT or specifies it as

zero in the initialization (HL01) transaction, then the HLAPI ignores the ALIAS_TABLE parameter and does not perform alias processing.

APPLICATION_ID
Contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses for this session. You can specify a value here to change the name of the current application ID. The application ID is specified on the HL01 transaction and can be specified on many other HLAPI transactions, so it can vary over the life of the HLAPI session. The ID must be an eligible user of the privilege class being used.

APPROVAL_STATUS
Contains the character value ACCEPT, which specifies that the specified privilege class is accepting the change. If you specify any other value, the change is rejected for the specified privilege class.

ASSOCIATED_DATA
Contains the symbolic name of the associated data field (PIRTDATA) item to be extracted from the record when processing a PIRT. The HLAPI requires this parameter if an inquiry transaction processes associated data. The value you specify in field PDBDATA must be a symbolic field index when you do not specify alias table processing, or an alias name when you specify alias table processing. If you specify a text or list item for this field, then the HLAPI does not return any associated data.

BEGINNING_HIT_NUMBER
Contains a 4-byte fixed field that specifies the beginning match number to return. If your application specifies zero, the HLAPI uses a value of one.

DATA_VIEW_NAME
Contains a character field that specifies a data view name either as an alias or data view record ID. You must specify an alias name as a 32-character left-justified field exactly as it appears in a PALT. You specify a data view record ID as a 1- to 8-character name. The API generates a PIDT from the data view record and associated data attribute and validation records. A PIDT shows the data view of the record being processed. All PIDTs and related PIPTs can be maintained in storage to improve performance. See *TABLE_COUNT* on page 155 for information on caching PIDTs.

Note: If a PIDT_NAME is specified as well, the PIDT_NAME is ignored.

DATE_FORMAT
Contains a character field that specifies how your application uses dates. Valid values are:

MM/DD/YY
MM/DD/YYYY
MM-DD-YY
MM-DD-YYYY
MM.DD.YY
MM.DD.YYYY
DD/MM/YY
DD/MM/YYYY
DD-MM-YY
DD-MM-YYYY
DD.MM.YY

DD.MM.YYYY
YY/MM/DD
YYYY/MM/DD
YY-MM-DD
YYYY-MM-DD
YY.MM.DD
YYYY.MM.DD
DDMMMYY
DDMMMYYYY
YYDDD
YYYYDDD

Note: The value DATABASE can also be specified; use of this value sets the format back to the default (database format); this is analogous to the PICADFMT=0 setting in the LLAPI (see page 20).

DEFAULT_OPTION
Contains a character field that specifies how the HLAPI performs create default data response processing in this session. The valid data values for DEFAULT_OPTION are ALL, REQUIRED, and NONE. ALL specifies that all response fields specified in a PIDT are candidates for default responses. REQUIRED specifies that only required fields are candidates for default responses. The HLAPI does not perform default response processing if you omit this field, specify it incorrectly, or specify it as NONE. You can override the initial default processing option when creating records by respecifying the default option on the control chain. After the create transaction completes, the HLAPI reverts to the initial default specification for record creation unless overridden again.

DELETE_HISTORY
Contains a character string specifying the date in external format of the oldest history data to be kept with the record. Any history created earlier than this date is deleted from the record. When a record is updated and the DELETE_HISTORY PDB is specified with a date value, the last saved PIHT is attached to the record and all history entries recorded before the given date are marked for deletion. The LLAPI checks that the correct PIHT (correct meaning that the PIHT was saved for the same record that the update is for) is attached to the record and deletes the marked entries. In the LLAPI that is shipped with the Tivoli Information Management for z/OS product, the history data update function is shipped disabled. Once enabled, either in TSP BLGAPI05 for panel processing or in TSP BLGAPIPX when bypassing panel processing, the user must have database administrator authority to successfully execute this transaction. Before data can be deleted, it must have been saved by setting the HISTORY_DATA PDB to S or B on a previous retrieve of the same record.

EQUAL_SIGN_PROCESSING
Contains the character value YES, which specifies that the HLAPI is to use equal sign processing. If you specify any other value, the HLAPI performs no equal sign processing.

HISTORY_DATA
Contains R, S, or B.

- R specifies that the HLAPI is to return at the end of the output PDB chain all of the history data contained in the record.
- S specifies that the HLAPI will save the PIHT retrieved with this record for later use on an update transaction. Any previously saved and unused PIHT is replaced.
- B specifies that the HLAPI performs both functions of R and S.

If you specify any other character or you omit this parameter, then the HLAPI does not retrieve or save history data for the record. No additional level of authority is required for this function beyond that for record retrieval. As with any PDB data returned on the output chain, the storage is freed on the next invocation of the HLAPI. Only one PIHT can be saved at a time, regardless of record type. The history saved from one record remains available for use until it is replaced by the saved PIHT of another record or until it is actually used. Subsequent transactions that do not save or use history data, retrieve or otherwise, have no effect on the saved history data.

LIST_MODE
Contains an option parameter UPDATE, APPEND, or REPLACE.

- A value of UPDATE specifies that any new list data input on the update will update existing lists in the record.
- A value of APPEND specifies that any new list data input on the update will be appended to the end of existing lists in the record.
- A value of REPLACE specifies that any new list data input will replace existing lists in the record.

If you specify any other value or if you omit this parameter, then the default value of UPDATE is used.

NUMBER_OF_HITS
Contains a 4-byte fixed field that specifies the maximum number of matches to be returned from a search:

- If this field contains a value, the actual number of matches is the smaller of:
  - The value in this field
  - The actual number of matches
  - The value in SORTPFX-N1.

PIDT_NAME
Contains a character field that specifies a PIDT name either as an alias or a member name. You must specify an alias name as a 32-character left-justified field exactly as it appears in a PALT. You specify a member name as a 1- to 7-character member name that is retrieved from the BLGFMT report format table data set. A PIDT shows the data view of the record being processed. All PIDTs and related PIPTs can be maintained in storage to improve performance. See *TABLE_COUNT* on page 155 for information on caching PIDTs.

Note: If a DATA_VIEW_NAME is specified, the PIDT_NAME is ignored.

PRIVILEGE_CLASS
Contains a 1- to 8- byte privilege class name, which can contain DBCS characters enclosed by an SO/SI pair. A privilege class remains in effect until your application specifies a different privilege class name. An application can

specify an initial privilege class that grants all authority required for the duration of the Tivoli Information Management for z/OS session.

REPLACE_TEXT_DATA
Can contain the character value YES which indicates that any text data provided is used to replace existing text of the same type. If any other value is specified, the text data provided is appended to any existing text of the same type. If the input consists of a single separator character, the existing text data is deleted. For deleting freeform text using buffer processing, both PDBDATL and PDBDATW must be set to the value 1.

RETURN_VALIDATION_DATA
Contains the character values YES or NO to indicate whether to return validation pattern data. The default is YES.

RNID_SYMBOL
Contains a 1- to 8-character external record identifier of the record to be processed. If the external record identifier is user defined, it can contain DBCS characters enclosed by an SO/SI pair.

SEARCH_ID
A 4-byte fixed field containing the identifier of a search. It is assigned to either a new search results list or an existing search results list. If the value of this field is zero, the search results are not saved.

SEARCH_TYPE
Contains a 1-byte character field indicating how the HLAPI treats this search. If this field is blank or set to S, it indicates to the HLAPI to start a new search. If this field is set to T, it indicates to the HLAPI to terminate an existing search. If this field is set to R, it indicates to the HLAPI to return matches from a saved search.

SEPARATOR_CHARACTER
Contains a 1-byte separator character the HLAPI uses to process response data. The value of the separator character, unless it is blank, overrides the value specified in the PIDT when this PDB is used for input transaction processing. The separator character must not be an SO or SI character. For a retrieve transaction, the value in PIDTSEPC (a comma by default) is used as the separator character.

TEXT_AREA
Contains a 1-character field indicating whether the HLAPI processes the bottom block or top block of text lines when the number of text units (lines) available exceeds the value of TEXT_UNITS. The valid values are T for the top block of text and B for the bottom block of text. If you specify a value other than T or B, or omit this parameter, then the HLAPI uses the default value B.

TEXT_AUDIT_OPTION
Contains the character value NO, which specifies that the HLAPI should not return audit data when retrieving text. This parameter has no meaning when creating or updating records. If you specify any other value or omit this parameter, and have requested text to be retrieved, the HLAPI returns audit data.

TEXT_DDNAME
Contains a 6-character field that specifies the prefix of the DDNAME (the leading 6 characters) that the HLAPI uses to generate application-defined

DDNAMEs for text data sets. You use this control field only when TEXT_MEDIUM=D and TEXT_OPTION=YES. As the HLAPI retrieves each text item, it allocates a data set using a DDNAME.

The DDNAME that the application generates for this data set contains the prefix characters that are specified by this parameter followed by a numerical ending that the HLAPI generates. For example, *ddname01, ddname02, ddname03*. The HLAPI can generate a maximum of 99 DDNAMEs.

The HLAPI passes text DDNAMEs to your application in output PDBs when retrieving record data. This allows your application to open the data set by way of the DDNAME so your application can process the record data immediately.

TEXT_MEDIUM
Contains the character B or D, which specifies the storage medium the HLAPI uses to store or return text data when retrieving records. The character D specifies that the HLAPI uses a data set or data sets, and the character B specifies that the HLAPI uses a storage buffer. If you specify any other character, then the HLAPI uses the default value of D. You use this parameter with other text processing parameters.

TEXT_OPTION
Contains the character value YES, which specifies that the HLAPI is to process text data items when retrieving records. This parameter has no meaning when creating or updating records. If you specify any other value, the HLAPI performs no text processing. You use this parameter with other text processing parameters.

TEXT_STREAM
Contains the character value YES, which specifies that freeform text is processed as a continuous stream of data which may include line feed characters. If you specify any value other than YES or omit this parameter, freeform text is processed as a series of fixed-width lines. This parameter may be specified on a retrieve, a create, or an update transaction.

TEXT_UNITS
Contains a 4-byte fixed binary value that specifies the maximum number of text units (lines) that the HLAPI can store in the response buffer for a single text item when retrieving records. You use this parameter with the TEXT_AREA parameter. If you specify a value of 0 or omit this parameter, and have requested buffer text processing, then the HLAPI uses a default value of 60.

TEXT_WIDTH
Contains a 4-byte fixed binary value that specifies the maximum width of a text unit (line) that the HLAPI can store in the response buffer. You use this parameter only with record retrieval transactions and with the TEXT_MEDIUM parameter. If you specify a value of 0 or a value greater than 132, or you omit this parameter, and you have requested buffer text processing (TEXT_MEDIUM=B), then the HLAPI uses a default of 60. Text width can be any value between 1 and 132.

TIME_ZONE
Contains a 1– to 8–character value that must match one of the values in the TIMEZONE record in the database. To use a time zone other than specified in the session parameters member, pass the TIME_ZONE control PDB on any HLAPI transaction with the desired TIMEZONE label as the data.

TRANSACTION_ID
Contains a 4-character transaction ID. Transaction IDs start with the letters HL followed by two numeric characters between 0 and 9.

You specify the following control parameters only when requesting an Initialize Tivoli Information Management for z/OS (HL01) transaction.

APIMSG_OPTION
Contains a 1-character LLAPI message option parameter P, C, or B.

- A value of P specifies that the LLAPI writes messages to the APIPRINT data set.
- A value of C specifies that the LLAPI chains messages and passes them from the LLAPI to the HLAPI for conversion into message PDBs.
- A value of B specifies that the LLAPI performs both P and C.

If you specify any other character or if you omit this parameter, then the LLAPI performs option C. This PDB is used only if SPOOL_INTERVAL is specified and is not set to zero.

BYPASS_PANEL_PROCESSING
Bypass panel processing indicator. Set this to **YES** to specify that no panels be used in record processing other than those used by the delete transaction. If you specify any other value, the HLAPI performs panel processing.

If you specify BYPASS_PANEL_PROCESSING = YES, you must use data model records for the following transactions:

- HL08 Create record
- HL09 Update record
- HL12 Add record relation

Additional information on BYPASS_PANEL_PROCESSING can be found in "API Control Flow" on page 283.

CICS_CM_TIME_OUT_VALUE
This PDB is used only in HLAPI/CICS client application programs. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for a description.

CICS_INTER_TIME_OUT_VALUE
This PDB is used only in HLAPI/CICS client application programs. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for a description.

CICS_PARTNER_ID
This PDB is used only in HLAPI/CICS client application programs. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for a description.

CICS_USER_ID
This PDB is used only in HLAPI/CICS client application programs. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for a description.

CLASS_COUNT
Contains a 4-byte fixed value that indicates the maximum number of Tivoli Information Management for z/OS privilege class records that can be maintained in storage during the life of this Tivoli Information Management for z/OS session. If you omit this parameter or enter zero as its value, the Tivoli Information Management for z/OS session operates with a single privilege class record in storage at a time.

DATABASE_ID
A 1-byte fixed field containing a 1-character ID number of the database to be used. For Tivoli Information Management for z/OS records, the database ID number is 5. If you omit this parameter, the HLAPI automatically sets the database ID to 5.

DATABASE_PROFILE
This PDB is used only in HLAPI/2, HLAPI/UNIX, and HLAPI/NT client application programs. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for descriptions.

DEFAULT_DATA_STORAGE_SIZE
Contains a 4-byte fixed value specifying how much additional storage is allocated to hold default response data from an alias table when your application is creating records. When the HLAPI creates records, it calculates the size of the response buffer it needs by totaling the lengths of all the input data PDBs and adding the specified default data storage size. If you omit the default data storage size, the HLAPI adds a default of 1024 bytes. When the HLAPI performs create response processing, it always checks to make sure the response will not overlay storage. If the response will overlay storage, the HLAPI transaction will end with an error code. You use this parameter with the DEFAULT_OPTION parameter.

HIGH_MEMORY
Contains the character value YES, which specifies that the HLAPI may return output, message, and error PDBs in memory that was obtained above the 16MB address range. If you specify any other value, these PDBs are always returned in memory obtained below the 16MB address range. If you are using the HLAPI through a remote client, do not use this PDB, because the value YES is always assumed.

HLAPILOG_ID
Must contain a 1- to 8-character HLAPI session identifier that you can specify to identify the session in HLAPI log file messages. If you do not specify a value for HLAPILOG_ID, then this field is blank in HLAPI log file messages.

HLIMSG_OPTION
Contains a 1-character HLAPI message option parameter P, C, or B.

- A value of P specifies that the HLAPI writes messages to the HLAPILOG data set.
- A value of C specifies that the HLAPI chains messages on the PDB message chain.
- A value of B specifies that the HLAPI performs both P and C.

If you specify any other character or you omit this parameter, then the HLAPI performs option C. The HLAPI writes messages passed back from the LLAPI to the HLAPILOG data set. This PDB is used only if SPOOL_INTERVAL is specified and is not set to zero.

MULTIPLE_RESPONSE_FORMAT
Contains a character value to indicate whether multiple response fields are separated by spaces or by the value specified for SEPARATOR_CHARACTER. A value of PHRASE specifies that each word in a multiple response is separated by a blank. If the value SEPARATOR or any value other than PHRASE is specified, then the words in a multiple response field are separated by the value specified for SEPARATOR_CHARACTER. The default is SEPARATOR.

PASSWORD
This PDB is used only in HLAPI/2, HLAPI/UNIX, and HLAPI/NT client application programs. Refer to the *Tivoli Information Management for z/OS Client Installation and User's Guide* for additional information.

SECURITY_ID
This PDB is used only in HLAPI/2, HLAPI/UNIX, and HLAPI/NT client application programs. The *Tivoli Information Management for z/OS Client Installation and User's Guide* contains additional information.

SESSION_MEMBER
Contains a 7- or 8-character load library session parameter member name that Tivoli Information Management for z/OS uses for this session. Session member names begin with the character string BLGSES and cannot contain imbedded blanks.

SPOOL_INTERVAL
Contains a 4-byte fixed value specifying the number of minutes that the HLAPI spools the activity logs HLAPILOG and APIPRINT when messages are printed. If the HLAPI is spooling to a data set and this time interval has passed, the activity logs are recycled and new log information is written starting at the top of the data set, writing over any existing information. If you omit this parameter, the HLAPI does not log messages and the settings in APIMSG_OPTION and HLIMSG_OPTION are ignored.

TABLE_COUNT
Contains a 4-byte fixed binary field value that indicates the maximum number of alias tables and PIDTs and anchored PIPTs that the HLAPI can maintain in storage during the life of a Tivoli Information Management for z/OS session.

Note: The maximum value that can be specified is 256.
Static PIDTs and PIDTs generated from data view records are treated the same for caching purposes. It can take a significant amount of time to generate a PIDT from data view records. The length of time depends on the number of data attribute records (and validation records they reference) contained in the data view record. Therefore, it can be especially important to direct the HLAPI to maintain PIDTs in storage if you are using data models. If you specify this value as zero or omit it, the Tivoli Information Management for z/OS session will not process ALIAS_TABLE parameters or cache PIDTs. Alias table and PIDT processing can increase transaction run time due to the increased I/O time of

loading and unloading tables. By balancing the table count to alias table and PIDT usage, you can reduce to zero the additional I/O overhead for long-running applications.

TIMEOUT_INTERVAL
Contains a 4-byte fixed value specifying the number of seconds that a transaction can run before a timer interrupt occurs. If you specify a value between 0 and 45 seconds, the HLAPI uses a value of 45 seconds. If you specify a value of 0 or omit this parameter, the HLAPI uses a default value of 300 seconds (five minutes).

INPUT chain
Specifies the input PDB parameters that certain transactions require for defining the data given to the HLAPI. The type of input data you define on the PDB chain is unique to the transaction being performed. The HLAPI anchors input PDBs to HICA field HICAINPP.

Your application must specify input response data in uppercase for consistency with interactive responses.

Your application stores list and multiple data responses for any single item in field PDBDATA of a single PDB. The format of list and multiple response data in PDBDATA is the same as required by the LLAPI when processing a response buffer. See “List Item Processing Considerations” on page 185 for more information.

For direct-add and phrase items, the field PDBDATA must contain any nonblank character to allow item collection. For text items, the parameter data specifies either the fully qualified data set name that is associated with the item or the actual text data. For the HLAPI to process this data set, the data set must not be currently allocated by the application. The HLAPI allocates this data set with a disposition of DISP=(SHR,KEEP,KEEP).

If the PDB refers to actual text data and control PDB TEXT_STREAM is not YES, field PDBDATW must contain the width of the text unit (line). PDBDATL must be an even multiple of field PDBDATW. There can be no remainder when dividing PDBDATL by PDBDATW. PDBDATW cannot be larger than 132 bytes. When you are in update mode and specify a new text width for freeform text, the width of the old data in the record remains unchanged. If the PDB refers to actual text data, and control PDB TEXT_STREAM is YES, PDBDATW and PDBDATL must both equal the total length of the text data. PDBDATW can be larger than 132 bytes.

Values for PDBNAME that have special meaning when used with specific HLAPI transactions are described below:

RETRIEVE_ITEM
Contains a 1- to 32-character field name that you want the HLAPI to retrieve. The specified name is the internal symbolic name or an alias name. See “Retrieve Record (HL06)” on page 171 for more information.

TSP_NAME
Contains a 1- to 8-character name of a TSP or TSX to invoke. If not specified, you must define a link to a TSP or TSX to invoke in either TSP BLGAPI00 or BLGAPIDI. See “Start User TSP or TSX (HL14)” on page 166 for more information.

USE_AS_IS_ARGUMENT
Contains a 1- to 33-character data argument that you can specify for the record inquiry transaction. See “Record Inquiry (HL11)” on page 194 for more information.

USER_PARAMETER
Contains a 4-byte address that you specify. The address points to a user-defined area that the HLAPI passes to a user TSP. See “Start User TSP or TSX (HL14)” on page 166 for more information.

USER_PARAMETER_DATA
Contains a 1- to 255-character string to be passed to the invoked TSP or TSX named in the PDB TSP_NAME. If TSP_NAME specifies a TSP to be invoked, the data is passed in the variable data area. If TSP_NAME specifies a TSX to be invoked, the data is passed as an argument to the TSX. If both USER_PARAMETER_DATA and USER_PARAMETER are specified, the value for USER_PARAMETER_DATA is used. See “Start User TSP or TSX (HL14)” on page 166 for more information.

OUTPUT chain
Specifies the output PDB parameter data that HLAPI produces for certain transactions. The type of output data the HLAPI defines on the PDB chain is unique to the transaction being performed. The HLAPI anchors output PDBs to HICA field HICAOUTP.

The HLAPI stores list and multiple data responses for any single item in field PDBDATA of a single PDB. The format of the data in PDBDATA is the same as is stored in the response buffer by the LLAPI. For direct-add and phrase items, the field PDBDATA contains the direct-add data or the visible phrase. For text items, the parameter data specifies either the data set name attributes *ddname.dsname* of the text data set that is associated with the text item, or the actual text data of the text item.

By providing the data set name attributes, your application can open the data set using the DDNAME in the same session. The HLAPI allocates this data set with a disposition of DISP=(NEW,DELETE,DELETE). You must start a free text data set (HL15) or delete text data set (HL16) transaction after processing the text data set to release HLAPI resources associated with this data set.

If the PDB refers to the actual text data, and control PDB TEXT_STREAM is not YES, field PDBDATW contains the width of a text unit (line) and PDBDATL contains the total text length including audit data, if audit data is requested. A description of audit data can be found 22. The number of text units can be calculated by dividing PDBDATL by PDBDATW. Each PDB that is returned by the inquiry transaction is symbolically named INQUIRY_RESULT. If the PDB refers to actual text data, and control PDB TEXT_STREAM is YES, PDBDATW and PDBDATL both equal the total length of the text data. PDBDATW can be larger than 132 bytes.

History data can be retrieved on a Record Retrieve transaction. Each history data item is returned in a separate output PDB after all other record data output PDBs.

MESSAGES chain
Specifies the message PDB parameter data that the HLAPI might produce as a result

of a transaction. Each PDB that references message data is symbolically named MESSAGE_DATA. The HLAPI anchors MESSAGE PDBs to HICA field HICAMSGP.

The HLAPI converts messages from the LLAPI message chain.

ERROR CODES chain

Specifies PIDT field error codes that the HLAPI detects as a result of being set by the LLAPI when the LLAPI finds errors with input or output data. These PDBs provide additional information that aid in debugging transaction errors. Each PDB on this chain references a PIDT entry and its associated error code value found in PIDT field PIDTCODE. The HLAPI identifies the PIDT entry by its PIDTSYMB value and performs no alias table conversion even if you specified alias table processing. The HLAPI anchors error PDBs to HICA field HICAERRP.

The HLAPI also stores field validation error codes in the chain when PDBPROC contains the character V and the field validation module BLGPPVFM detects a pattern error.

See page 123 for descriptions of PIDT codes 00 through 46. The other field validation error codes specific to the HLAPI are as follows:

50 Data does not match any validation patterns.
51 Field symbolic name was not found in PIDT.
52 PIPT structure is not valid.
53 PIDT structure is not valid.
54 PIDT contains zero entries.
55 Field symbolic name was not found in PIPT.
56 PIDT contains a zero value in field PIDTFPAT.
57 Length of data to verify is zero.
58 Pointer to the data to be verified contains zero.
59 An unknown pattern validation character was found.
60 An R or V value is too large in a pattern.
61 No R or V value was found in a pattern.
62 A literal pattern does not end with a >.
63 An R or V value is too small in a pattern.
64 An imbedded blank was found in the data.
70 An unknown validation data type was encountered.
71 A BLX internal logic error has occurred during the processing of mixed data.
72 Field contains incorrect mixed data.

Data Model Information

This is the control block HIDM, which maps the output PDB field PDBDATA when the PDBTYPE field=F for data model information. Most of these map to fields of the PIDT, described in Table 33 on page 116. In addition to those that map to the PIDT, some additional fields are contained in the HIDM:

- HIDMXPMT – Default Prompt for Attribute (from the data attribute record)
- HIDMXHTH – HTML Help File Name (from the data attribute record)
- HIDMXJPX – Desktop Program Exit Name (from the data attribute record)
- HIDMXATS – Associated TSP/TSX Name (from the data attribute record)
- HIDMXAUT – Field Authorization Code (from the data view record)
- HIDMXVSX – Validation Field S-word Index (from the data attribute record)

- HIDMXRPY – Reply Is Always Data (from the data attribute record)

Note: The value for HIDMREQD is always returned as N.

Table 70. HIDM—Data Model Information Control Block. Maps the contents of the output PDB field PDBDATA when the PDBTYPE field=F for data model information. (Field Labels containing an asterisk * are reserved.)

Field Label	Offset DEC (HEX)	Length
HIDMSYMB	0(0)	5
HIDMRDEF	5(5)	1
*	6(6)	2
HIDMMNCR	8(8)	4
HIDMMAXL	12(C)	8
HIDMREQD	20(14)	1
HIDMDATE	21(15)	1
HIDMSRCH	22(16)	1
HIDMJRNL	23(17)	1
HIDMLIST	24(18)	1
HIDMRTYP	25(19)	1
HIDMFAUP	26(1A)	1
HIDMSDAT	27(1B)	1
HIDMLZPD	28(1C)	1
HIDMPNLN	29(1D)	8
HIDMPFXD	37(25)	6
HIDMVISD	43(2B)	28
HIDMCSVL	71(47)	1
HIDMCGMX	72(48)	1
HIDMCDCA	73(49)	1
HIDMXPMT	74(4A)	25
*	99(63)	3
HIDMXHTH	102(66)	12
HIDMXJPX	114(72)	8
HIDMXATS	122(7A)	8
HIDMXAUT	130(82)	4
HIDMXVSX	134(86)	4
HIDMXRPY	138(8A)	1
*	139(8B)	21

Data Model Validation Pattern Data

This is the control block HIVP, which maps the output PDB field PDBDATA when the PDBTYPE field=V for data validation data. These map to fields of the PIPT, described in Table 39 on page 136.

Table 71. HIVP—Validation Pattern Data Control Block. Maps the contents of the output PDB field PDBDATA when the PDBTYPE field=V for data validation data.

Field Label	Offset DEC (HEX)	Length
HIVPSYMB	0(0)	5
HIVPTYP	5(5)	1
HIVPAUTH	6(6)	4
HIVPRSV3	10(A)	8
HIVPPRFX	18(12)	6
HIVPDATA	24(18)	32
*	56(38)	24

Alias Tables

Alias tables are table structures that reside as members of a partitioned data set. This partitioned data set must be a member of the Report Format Table data set concatenation. You use alias tables to specify alias names for Tivoli Information Management for z/OS p-word and s-word indexes, and to create default response values used when response data is not specified in create and update record transactions. You can also use these tables to specify alias names for the RFT data set member name for a PIDT. Alias tables are constructed using the Table Build Utility BLGUT8. The Table Build Utility lets you use any valid PDS member name for alias tables.

Alias table processing is not performed when processing a control PDB. The HLAPI processes control PDBs using reserved names.

Alias tables (see Table 72 on page 239) contain five data columns in the following order:

1. **Internal symbol** column containing up to 5 characters. Internal symbols are left justified and padded with blanks. They can be s-word indexes. They cannot be blank or contain imbedded blanks. These values correspond to the values in the PIDTSYMB field.
2. **Alias value** column containing up to 32 characters. Alias values are left-justified uppercase character strings padded with blanks containing only the characters A-Z, 0-9, underscore, and period. They cannot be blank or contain imbedded blanks.
3. **Default response data** column containing up to 45 characters. Default response data fields are left-justified uppercase character strings padded with blanks. They can be blank and can contain imbedded blanks, and may also contain mixed data containing DBCS characters enclosed by an SO/SI pair. The HLAPI processes these responses, but it does not validate them.
4. **Tivoli Information Management for z/OS p-word value** column containing up to 6 characters left justified and right padded with blanks. They can be p-word indexes or p-words. Your application uses p-words to construct inquiry arguments.
5. **PIDT member name value** column containing up to eight characters left-justified and padded with blanks. Member name values are static PIDT names or data view record IDs.

This table summarizes the alias table format.

Table 72. Alias Table Format

Internal Symbol	Alias value	Default Response	P-Word	PIDT Member
5 characters	32 characters	45 characters	6 characters	8 characters

Entry INPUT Name Processing

The HLAPI processes the alias table from top to bottom, locating the first alias name that matches the name specified in PDB field PDBNAME. When the HLAPI finds a match, it verifies that the corresponding internal name is an s-word index or a p-word index and that the name exists as a PIDT symbolic name within the PIDT specified for use. The HLAPI then processes the PDB. If the HLAPI cannot find the name in the alias table and it appears to be a valid s-word or p-word index, then the HLAPI processes the name as an internal symbolic name. If the HLAPI cannot find a name match in the PIDT, the input PDB's PDBCODE is marked with an error code M, and the transaction terminates with an error.

Default response processing attempts to extract responses from the alias table when you do not specify input PDB responses, and DEFAULT_OPTION is other than NONE. If the HLAPI cannot find a default response for a required field, the transaction ends with an error.

Retrieve INPUT Name Processing

To perform retrieve INPUT processing, your application must specify the name of the data item to be retrieved in PDB field PDBDATA and reserved name RETRIEVE_ITEM in field PDBNAME. If you specify a field that contains no data in the record, the API sets PDBCODE to E, indicating that the field is null. If you specify a field that cannot be found in the PIDT or PALT, the API sets PDBCODE to M, indicating that the field is undefined and no match is found to process.

Inquiry INPUT Name Processing

Your application can perform inquiry structured search processing by specifying the alias of an s-word index or p-word index in PDB field PDBNAME and specifying the argument data in PDBDATA. The HLAPI matches the alias name you specify with an alias table row and extracts the internal symbol from that alias table row. If the HLAPI does not find a match in the alias table, and the name appears to be a valid s-word or p-word index, the HLAPI processes the name as a PIDT symbolic name. If this PIDT symbolic name does not match the identifier of a PIDT row (PIDTSYMB), the transaction ends with an error.

Inquiry freeform search processing requires that the name specified in PDB field PDBNAME be an alias of a p-word or the reserved name USE_AS_IS_ARGUMENT. If the HLAPI finds a p-word alias, the HLAPI stores the p-word from the p-word value field of the alias table in the argument first. This is followed by the data specified in PDBDATA. If the HLAPI does not find a p-word alias, the transaction terminates with an error. When your application specifies PDBNAME as USE_AS_IS_ARGUMENT, the HLAPI stores just the argument, as specified in PDBDATA.

Record Retrieve OUTPUT Name Processing

The HLAPI attempts to locate the first match between the PIDT symbolic name PIDTSYMB and an alias table internal name. If the HLAPI finds a match, it extracts the external name and stores it in PDB field PDBNAME. If the HLAPI cannot find an internal name, the

HLAPI stores the PIDT symbolic name. When performing RETRIEVE_ITEM processing, the name stored in the RETRIEVE_ITEM PDB's PDBDATA field is used to name each found output PDB.

Using the HLAPI/REXX Interface

The High-Level Application Program Interface/REXX (HLAPI/REXX interface) enables you to access HLAPI transactions from REXX programs. You can write a REXX program that sets variables with control and input information and then links to the HLAPI through the HLAPI/REXX interface to process that information. On return, the HLAPI/REXX interface uses data returned by the HLAPI to set various REXX variables in your program. All the HLAPI transactions are available to the HLAPI/REXX interface. For a list of those transactions, see Table 73 on page 242.

To use the HLAPI/REXX interface, certain information is required. Be sure to specify the name of the transaction that you want to perform. This is a parameter to the HLAPI/REXX interface module. It is converted into the HLAPI TRANSACTION_ID PDB. You must know static PIDT names or data view record IDs for each transaction that requires a PIDT, for example, create, update, or retrieve. You must specify the control information that allows the HLAPI to perform the transaction, and, in many cases, you must specify input information to give the HLAPI the necessary data to use in its transaction. Examples of input information are:

- Structured or freeform search arguments for use in inquiries
- Data items for use in create record or update record transactions
- Data item names for fields you request in a record retrieve transaction.

You set all of this data in the variables of your REXX program, which then calls the HLAPI/REXX interface.

The HLAPI/REXX interface uses the REXX variable names and values that you set to create PDBs for input to the HLAPI. Upon return, output from the HLAPI, such as return codes, record IDs, and error messages, is used to set other REXX variables. The HLAPI/REXX interface does not interpret inputs or results, it acts as an intermediary between REXX programs and the HLAPI. In general, your REXX program must provide all the control and input information that the HLAPI and its transactions require. But, if you specify control information for a transaction that is not required or used by the HLAPI transaction, the specified control information is ignored. Therefore, when you use the HLAPI/REXX interface, refer to "HLAPI Transactions" on page 151 for a description of the control and input information that each transaction requires.

The HLAPI/REXX interface environment is stored in the REXX program by the interface when you initialize a session. The environment is stored in REXX variable BLG_ENVP. This variable must be present in the REXX program for all calls made to the HLAPI/REXX interface after you initialize a session. When your program ends a session, BLG_ENVP is dropped.

Like the HLAPI, the HLAPI/REXX interface can use alias tables. Also, logging of messages for the HLAPI/REXX interface works like it does for the HLAPI. HLAPI message PDBs are written to the BLG_MSGS array, and the HLAPI controls logging to HLAPILOG. If HLIMSG_OPTION and APIMSG_OPTION are both set to either C or B, the HLAPI/REXX interface receives messages on the message chain. If HLIMSG_OPTION and APIMSG_OPTION are both set to P, the HLAPI/REXX interface receives no messages.

Date Considerations

Dates used by your application can be processed in either of two ways:

Database format

Dates are passed to your application from the API in the default external date format. Dates your application passes to the API must be in either the default format or, if one is defined, the old format specified in the session parameters being used. Dates passed in either format are automatically converted to internal format when they are stored in the SDDS portion of the database.

Application-specified format

Dates are passed between the API and your application in a date format your application specifies. This format does not need to match that of the database. The API automatically converts dates from the internal format in the database to the format you specify when passing data to your application and from your specified format to the database's internal format when receiving data from your application.

An application-specified date format is set in the HLAPI/REXX by specifying the desired date format (for example, MM/DD/YYYY or YYYY.MM.DD) in a control data item with a name of DATE_FORMAT.

Database date format is the default and can be specified in the HLAPI by never specifying a control data item named DATE_FORMAT.

Differences between the HLAPI/REXX Interface and the HLAPI

Although the HLAPI/REXX interface depends on the HLAPI, some differences are found between the two in such things as how they process inputs, what inputs they expect, and how information is returned to the user.

Some of the significant differences between the HLAPI/REXX interface and the HLAPI are:

- Many transactions of the HLAPI require that you specify a separator character. The HLAPI/REXX interface does not require a separator character. It uses a comma as the default separator character if you do not specify one.
- When using the HLAPI/REXX interface, you do not specify the transaction ID as part of the control chain. You specify the transaction as the first parameter to the HLAPI/REXX interface. So, in some cases where the HLAPI requires a control chain, the HLAPI/REXX interface does not.
- You can specify PDB data tracing in both cases. This enables you to trace the PDBs that are input to and output from each HLAPI transaction. Trace information is sent to the HLAPILOG data set. The HLAPI initialize Tivoli Information Management for z/OS transaction (HL01) requires you to set PDBPROC in the TRANSACTION_ID PDB to T before executing the transaction. The HLAPI/REXX interface requires the variable INIT.?PROC to be set to T before you request an INIT transaction.

HLAPI/REXX Interface Calls

The following example shows the syntax for interface calls. *Italics* indicate a REXX variable; you define its name in the REXX program. You must enter other key information exactly as shown. The brackets ([and]) indicate parameters that are optional for some transactions. In most cases, you must provide various control and input information. Output is always optional because the interface uses the variable BLG_OUT., if you do not specify an output stem.

```
ADDRESS LINK "BLGYRXM transaction-name[,control,input,output]"
```

transaction-name
Specifies the transaction to perform.

control
An optional item. It is the stem of a compound variable that identifies control data items that this transaction uses.

input Another optional item. It is the stem of a compound variable that identifies the input data to use when processing the transaction.

output An optional item. It identifies the stem of a compound variable for the HLAPI/REXX interface to use to return output data from the HLAPI.

Use an extra comma in the variable list to indicate that you are not passing a particular parameter. For example:

```
ADDRESS LINK "BLGYRXM RETRIEVE,control_list,,output_stem"
```

indicates that the third parameter, *input*, is not passed.

The load module BLGYRXM must be in one of the following areas:
- Job pack area
- Task library
- Step library
- Link Pack Area (LPA)
- Link library.

Note: When the Katakana configuration is used, any REXX program variable names that are related to *transaction name, control, input, output* must be entered in uppercase. Remember the Katakana code page does not include lowercase English letters.

Transaction Name

The first parameter to the HLAPI/REXX interface is the name of the transaction to perform. This is the only parameter required by all transactions. Names for each transaction and the corresponding HLAPI transaction code are listed in Table 73. If you are not using the Katakana configuration, you can pass the names of the transactions in lowercase, uppercase, or mixed case in your REXX program.

Table 73. HLAPI/REXX Transactions

Name	Number	Purpose	Page
INIT	HL01	Initialize Tivoli Information Management for z/OS Environment	153
TERM	HL02	Terminate Tivoli Information Management for z/OS Environment	160
GETID	HL03	Obtain External Record ID	161
CHECKOUT	HL04	Check Out Record	162
CHECKIN	HL05	Check In Record	164
RETRIEVE	HL06	Retrieve Record	171
CREATE	HL08	Create Record	178
UPDATE	HL09	Update Record	183
CHANGE_APPROVAL	HL10	Change Record Approval	226
SEARCH	HL11	Record Inquiry	194

Table 73. HLAPI/REXX Transactions (continued)

Name	Number	Purpose	Page
ADD_REL	HL12	Add Record Relations	202
DELETE	HL13	Delete Record	205
USERTSP	HL14	Start User TSP	166
FREE_TEXTDS	HL15	Free Text Data Set	169
DEL_TEXTDS	HL16	Delete Text Data Set	170
GETDATAMODEL	HL31	Get Data Model	207

Control Data

The HLAPI uses control information to set up the environment that it needs to perform a transaction or to provide information about how to process a transaction. You define a compound variable in your REXX program to contain the names of control data. The HLAPI/REXX interface reads this compound variable and the associated control data and passes it to the HLAPI as a control PDB chain.

Most transactions require specific control information. The control names that define this information are the same as the reserved alias names that the HLAPI uses for control items. One control name that the HLAPI normally uses, but that is not used by the HLAPI/REXX interface, is the requested transaction control name (TRANSACTION_ID). For the HLAPI/REXX interface, the transaction requested is passed as a parameter. Also, the SEPARATOR_CHARACTER control value defaults to a comma if you do not define one. For a description of the control items see “Parameter Data Definition” on page 225.

Your REXX program must provide all of the required control information and any optional control information necessary to perform the transaction you want. You can find information concerning HLAPI transactions and the required and optional control information for each in “HLAPI Transactions” on page 151.

Once you have set the variables containing the control information, build a compound variable that defines which control variables you want to use for a transaction. The stem of the compound variable cannot exceed 32 characters. Set each element value to a control name, for example, create_cntl.1='ALIAS_TABLE'. The .0 element of the compound variable must be set to the number of the largest element of the compound variable. This number cannot be greater than 30.

The stem of the control compound variable is passed as a parameter to the HLAPI/REXX interface. The HLAPI/REXX interface reads the .0 element, which determines the maximum element number to read. The HLAPI/REXX interface then reads all elements with tails up to that number. For example, if the .0 element contains a value of 4, the HLAPI/REXX interface reads elements with tails of 1, 2, 3 and 4. Any undefined or null elements are ignored. This data tells the API which control variables to use for this transaction and the values of those control variables. In this way, you can define different *control* compound variables for each of the HLAPI/REXX interface transactions. If you have a series of transactions that you want to perform, such as create, retrieve, update, create, retrieve, update, you need not update the values of variables before every transaction. You can define different control variables and set them once for create, once for retrieve, and once for update. This is an example of how to define and specify control information for the HLAPI/REXX interface.

```
/*********************************************************************/
/* Partial REXX code to illustrate control information               */
/*********************************************************************/

/*********************************************************************/
/* Define control information for session.                           */
/*********************************************************************/

application_id='APPL01';
privilege_class='MASTER';                 /* initial class is MASTER */
session_member='BLGSES43';
table_count=5;                            /* this allows for 5 alias */
                                          /* tables                  */
hlimsg_option='C';                        /* return HLAPI messages as*/
                                          /* chain - returned to REXX*/
                                          /* exec as compound var    */
/*********************************************************************/
/* Specify which control variables to be used.  This                 */
/* information is given to the HLAPI and used to establish a         */
/* HLAPI session.                                                    */
/*********************************************************************/

init_control.1='privilege_class';
init_control.2='session_member';
init_control.3='application_id';
init_control.4='table_count';
init_control.5='hlimsg_option';
init_control.0=5;                         /* element 5 is the highest */
                                          /* element                  */
ADDRESS LINK "BLGYRXM INIT,init_control"
```

Input Data

Many transactions, such as create, update, and search, use input data. Types of inputs include:

- Entry items
- Structured search argument segments
- Freeform search arguments
- Input items that do not contain data, such as the names of items to retrieve from a record.

In all cases when the input items contain data, you perform two steps, in any order, to define your inputs as follows:

- Put the input data into REXX variables.
- Set up an array (or compound variable) to explicitly identify the names of the items you want to use in your transaction.

These two steps are discussed in detail below.

For items that do not have data, such as the names of fields to retrieve from a record, you identify the names of the items to use in the transaction. You enter freeform search arguments differently from all other types of inputs. Text items also require some unique rules to use when defining them as inputs. Both of these unusual cases are described in later sections of this chapter.

Put Data into Variables

All of the data you use with the HLAPI transaction must be placed into REXX variables. Ensure that you assign the correct length to your input variable data, and that the data does not contain leading or trailing blanks if they are not allowed. An easy way to remove leading and trailing blanks is to use the REXX function `STRIP`. For example, you want to create a record, and you assign the input data for the record ID using the statement:

```
input = '  12345   '
```

and you assign a record ID with the statement:

```
record_id = substr(input,3,8)
```

because the maximum value of a Tivoli Information Management for z/OS record ID is 8. The value of `record_id` is `'12345   '` after the last statement above runs. The trailing blanks do not pass validation when you attempt to create the record. To correct this problem use the following statement to assign a value to `record_id`:

```
record_id = strip(input,,' ')          /* Remove leading and trailing */
                                       /* blanks from input data, and */
                                       /* assign a value for the new  */
                                       /* record ID.                  */
```

The value of `record_id` after the above statement runs is `'12345'`, and the record create transaction passes validation.

Most Common Types of Input Data

Some examples of input values that are neither freeform text nor freeform search arguments are:

- Structured search arguments
- Data items used in a create or update transaction
- Text data set name or text data set DDNAME.

You can also perform text searching; this is described in "Text search argument inputs" on page 246.

The three elements of most input items are its name, its value, and a processing flag. Each item must have a name: the value is optional for some transactions, and the flag is optional for all transactions. A processing flag of V specifies that you want the HLAPI to perform automatic validation of your input. The name of the input must be either an alias name or the PIDTSYMB value (defined in the specified PIDT) for a field. You set a REXX variable corresponding to the alias name (if you use alias table processing) or PIDTSYMB value of an item with the data for that item. For example,

```
STATUS='OPEN'  /* alias table set up to make STATUS an alias */
               /* of S0BEE                                   */
        or
S0BEE='OPEN'   /*S0BEE is the PIDTSYMB value for STATUS      */
```

Note: Tivoli Information Management for z/OS non-freeform text data is always uppercase, so all data values must be uppercase; data is not converted.

Freeform search argument inputs

Freeform arguments can be part of the search argument used for a search transaction. Some types of freeform arguments are:

- Argument data used as specified with no prefix substitution
- Prefix alias name and data pairs
- Prefix index and data pairs.

The two elements of a freeform search argument input are the prefix alias or prefix index, and the data. You must specify the data, but the prefix alias or index is optional. The data can be mixed, which contains DBCS and SBCS characters.

You define the freeform search argument input by using a compound REXX variable with the stem `freeform`. You set `freeform.0` in the array to the number of freeform argument segments that you want to define. If you set `freeform.0` to zero or leave it undefined, no freeform arguments are processed. This number cannot be greater than 32 767. The other parts of the array are made up of two elements: `freeform.`*n*`.?prefix` and `freeform.`*n*`.?data`, where *n* is the number of the element. To specify a freeform argument segment that you want to use as is, which corresponds to the HLAPI PDBNAME USE_AS_IS_ARGUMENT, set `freeform.`*n*`.?data` to the argument data and leave `freeform.`*n*`.?prefix` undefined.

If you want to specify a freeform argument segment that uses prefix substitution, set `freeform.`*n*`.?prefix` to the prefix alias, and set `freeform.`*n*`.?data` to whatever data goes with it.

The data for each freeform argument segment can be up to 33 characters (32 characters for a search argument and one optional Boolean/range character), but it can consist of only one argument segment. The HLAPI combines each segment to make a multiple item search argument. Here is an example of defining freeform search arguments:

```
/*************************************************************/
/* Partial REXX code to illustrate defining freeform type search */
/* arguments.  Assume STATUS_PREFIX is defined in an alias table */
/* as prefix STAC/.                                          */
/*************************************************************/

freeform.1.?data='PERS/BILL';
freeform.2.?data='|PERS/WILLIAM';
freeform.3.?prefix='STATUS_PREFIX';
freeform.3.?data='OPEN';
freeform.0=3;
```

The resulting partial argument is:

```
PERS/BILL |PERS/WILLIAM STAC/OPEN
```

Note: You can use parentheses in your search argument to help narrow down search results. Arguments placed within parentheses will be evaluated first. For details on using parentheses in HLAPI search transactions, see 194.

Text search argument inputs

In order to do text searching using the HLAPI/REXX SEARCH transaction, you define the text argument inputs by using an array with one element for each text search argument. These can be simple text arguments, search phrases, or complex search arguments. The stem of the array must be *text*. This corresponds to the HLAPI PDBNAME TEXT_SEARCH_ARGUMENT. The reset of the array name is a number that indexes and defines the order of the text arguments. Set *text.0* to the number of text argument elements that you want to define. If you set *text.0* to zero or leave it undefined, no text arguments are processed. This number cannot be greater than 32 767.

You can specify the maximum width for your text arguments by letting *text.?width* to the width you want; however, this width cannot exceed 132 characters. If you do not specify a value for width, the HLAPI/REXX interface uses the default width of 60. The HLAPI combines the array elements to make a complete text search argument. This is an example of defining text search arguments:

```
/*************************************************************/
/* Partial TEXX code to illustrate defining text type search */
/* arguments.                                                */
```

```
/*************************************************************/
text.1='(solution AND "results ranking")'
text.2='OR syntax'
text.0=2
```

The resulting text argument is (solution AND “results ranking”) OR syntax

Defining Text Items

When specifying text, you have the same options with the HLAPI/REXX interface as you do with the HLAPI. You can specify a data set name or the text itself. You define the data set name like any other non-freeform search argument specification. See “Most Common Types of Input Data” on page 245. This part of the chapter explains how to define actual text for input. The text can be mixed data, containing DBCS and SBCS characters.

Set up an array with one element for each line of text you want to input. The name of the array (*text-name.n*) is user defined. The stem of the array (*text-name*) must be an alias name or the PIDTSYMB value that identifies what type of text you are using. The rest of the array name is a number that indexes the text lines or defines the order of the text lines. To specify a blank line of text, either skip an index, or set the array element to null.

Set the variable *text-name*.0 to the number of lines that you want to input. This number cannot exceed 32 767. The HLAPI/REXX interface reads array elements from *text-name*.1 to *text-name.n*, where *n* equals the value of *text-name*.0. If you do not specify *text-name*.0 or you give it a value of zero or null, then no text lines of this type are specified.

You can specify the maximum width for your text lines by setting *text-name*.?width to the width you want. If control PDB TEXT_STREAM is not YES, this width cannot exceed 132. If control PDB TEXT_STREAM is YES, this width should equal the length of the text and thus may be greater than 132. If you do not specify a value for width, the default width for the HLAPI/REXX interface is 60. The HLAPI/REXX interface truncates or pads each text line with blanks as necessary to meet the chosen width.

Here is an example of defining text items:

```
/*****************************************************************/
/* Partial REXX code to illustrate defining freeform text lines. */
/* Assumes DESCRIPTION_TEXT is defined in an alias table as      */
/* S0E01.                                                        */
/*****************************************************************/

DESCRIPTION_TEXT.1='This is a bad problem.';
DESCRIPTION_TEXT.2='System not responding.';
DESCRIPTION_TEXT.3='';
DESCRIPTION_TEXT.5='Slow down when you type.';
DESCRIPTION_TEXT.0=5;
```

After a successful create or update transaction, the resulting text looks like this on an interactive screen:

```
This is a bad problem.
System not responding.


Slow down when you type.
```

The third and fourth lines are blank because the .3 element is null and the .4 element is null.

Set Up the Array to Identify the Input

After you define the input data (if that step was necessary), you use a REXX array to explicitly identify which input items to use for a transaction. You also use this array to specify processing flags associated with input items. The variable is made of two parts—`input.n.?name`. and `input.n.?proc`. The stem `input` cannot exceed 32 characters.

Set `input.n.?name` to the name of the input item, for example `STATUS`, and set `input.n.?proc` to `V` if you want validation performed for this item. The names you use can be upper case or mixed case. If you want to specify an array of text, set `input.n.?name` to the stem of the text array including a period (.) at the end. This tells the HLAPI/REXX interface that this input is a text array. For example,

```
input.1.?name='DESCRIPTION_TEXT.'
```

You must set the variable `input.0` to the number of input data items you want to specify. The maximum number is 32 767 (entered without a comma or space, 32767). Freeform arguments and text arrays each count as one item.

You pass `input` as a parameter to the HLAPI/REXX interface. Each `input.n.?name` element identifies a single input except for two cases. The first exception is when you specify freeform. The freeform array identifies one or more freeform argument segments for a search. The second exception is when you specify aSTATUS text array. The array identifies one or more freeform text lines.

The HLAPI/REXX interface attempts to read `input.n.?name` and `input.n.?proc` for all `n`, from 1 to the value of `input.0`. If any `input.n.?name` values are not found or are null, they are ignored. However, if the name you specify in `input.n.?name` does not exist or has a null value, an error occurs.

When you do not want input data for a transaction, do not pass an input compound variable stem to the HLAPI/REXX interface. For example, if you specify `inptarry.5.?name='STATUS'` and you do not have the variable `STATUS` defined in your program, an error results.

Maximum Input Lengths

The maximum input lengths for values of HLAPI/REXX interface variables are defined in the following table:

Table 74. Maximum input lengths for interface variables

HLAPI/REXX interface variables	Maximum input lengths
control data values	32 characters
input processing flags	1 character
Control.0	2 numeric characters
Input.0	5 numeric characters
FREEFORM.0	5 numeric characters
Text-name.0	5 numeric characters
Text-name.?width	10 numeric characters
freeform argument segments	33 characters
numeric control data	10 characters
Text.0	5 numeric characters

Table 74. Maximum input lengths for interface variables (continued)

HLAPI/REXX interface variables	Maximum input lengths
Text.?width	10 numeric characters

Examples of Specifying Inputs

The examples in this section give you an idea of how to code inputs in REXX for create, search, and qualified retrieve transactions.

This is an example of how to code inputs in REXX for create transactions.

```
/*********************************************************************/
/* Partial REXX code to illustrate defining control and input        */
/* information for a record create.  Assume a session has            */
/* already been initialized.                                         */
/*********************************************************************/

/*********************************************************************/
/* Define control information.                                       */
/*********************************************************************/
pidt_name='BLGYPRC';                  /* problem create PIDT         */
separator_character=',';
alias_table='PROBAL';                 /* name of alias table to use  */

/*********************************************************************/
/* Specify control information to use.                               */
/*********************************************************************/
control.1='pidt_name';
control.2='separator_character';      /* defaults to comma if not    */
                                      /* specified                   */
control.3='alias_table';              /* use alias tableproc         */
control.0=3;

/*********************************************************************/
/* Set input variables - this is the data for the create.            */
/* Assume REPORTER_NAME has the value DOE/JOHN                       */
/*********************************************************************/
status='INITIAL';
description=REPORTER_NAME 'HAS A PROBLEM WITH HIS TERMINAL';
reporter_phone='555-1212';
system_name='PLL8772';
description_text.?width=60;           /* width of description text   */
                                      /* lines to be built - this is */
                                      /* the default                 */
description_text.2='PROBLEM REPORTED ON 01/21/1998.';
description_text.3='IT IS A VERY BAD PROBLEM.';
description_text.4='';
description_text.0=4;
status_text.1='THIS PROBLEM IS IN INITIAL STATUS.';
status_text.0=1;
status_text.?width=26;
/*********************************************************************/
/* Explicitly identify which input variables to use.                 */
/*********************************************************************/
myinput.1.?name='REPORTER_NAME';
myinput.1.?proc='V';                  /* tell HLAPI to validate the  */
                                      /* reporter name               */
myinput.2.?name='STATUS';
myinput.3.?name='DESCRIPTION';
myinput.4.?name='REPORTER_PHONE';
myinput.5.?name='SYSTEM_NAME';
myinput.6.?name='STATUS_TEXT.';       /* note-must specify '.'       */
myinput.7.?name='DESCRIPTION_TEXT.';  /* note-must specify '.'       */
myinput.0=7;                          /* element 7 is the highest    */
                                      /* element of the myinput array*/
```

```
/*********************************************************************/
/* call BLGYRXM.  Output information is returned in array            */
/* with stem 'outinf.'.                                              */
/*********************************************************************/
 ADDRESS LINK "BLGYRXM create,control,myinput,outinf";
```

This is an example of how to code inputs in REXX for search transactions.

```
/*********************************************************************/
/* Partial REXX code to illustrate defining control and input        */
/* information for a search.   Assume a session has                  */
/* already been initialized.                                         */
/*********************************************************************/

/*********************************************************************/
/* Define control information.                                       */
/*********************************************************************/
pidt_name='BLGYPRI';                  /* use problem inquiry PIDT    */
separator_character=',';
alias_table='PROBAL';                 /* name of alias table to use  */
/*********************************************************************/
/* Specify control information to use.                               */
/*********************************************************************/
control.1='pidt_name';
control.2='separator_character';      /* defaults to comma if not    */
                                      /* specified                   */
control.3='alias_table';              /* use alias table processing  */
control.0=3;

/*********************************************************************/
/* Define input data.  Assume you want to search on all problems     */
/* with:                                                             */
/* Status of OPEN                        and                         */
/* Date Opened of 01/21/1998             and                         */
/* Reporter name of BILL                 and                         */
/* any phone number starting with 555    and                         */
/* any phone number starting with 777    and                         */
/* TEST in any string field.                                         */
/*                                                                   */
/* Status and Date opened are structured arguments and the rest are  */
/* freeform arguments.  Assume that PROBAL alias table has STATUS de-*/
/* fined as s-word S0BEE, DATE_OPENED defined as s-word S0C3E, and   */
/* PHONE_PREFIX defined as prefix PH/  Because DATE_OPENED is a struc-*/
/* tured argument, you must specify the date in external format. If  */
/* you want to specify a date freeform argument you must do it using */
/* the internal format of YYYY/MM/DD.                                */
/*********************************************************************/
freeform.1.?data='PERS/BILL';
freeform.2.?prefix='PHONE_PREFIX';
freeform.2.?data='555.';
freeform.3.?prefix='PHONE_PREFIX';
freeform.3.?data='777.';
freeform.4.?data='TEST';
freeform.0=4;
STATUS='OPEN';
DATE_OPENED='01/21/98';
/*********************************************************************/
/* Explicitly identify which inputs to include.  Note that no matter */
/* what order they are specified the search argument always includes */
/* structured arguments first (using the order that the fields are de-*/
/* fined in the PIDT) and then the freeform arguments second.  The   */
/* freeform arguments are used in the order specified.               */
/*                                                                   */
/* The whole search argument becomes:                                */
/* status-s-word STAC/OPEN date-opened-s-word DATO/1998/01/21        */
/* PERS/BILL PH/555. PH/777. TEST                                    */
/*********************************************************************/
 srcharg.1.?name= 'FREEFORM';
 srcharg.2.?name= 'STATUS';
 srcharg.3.?name= 'DATE_OPENED';
 srcharg.0=3;

 /*********************************************************************/
 /* call BLGYRXM.  Search outputs will be returned into               */
```

```
 /* array with stem 'srchres.'.                                        */
 /*********************************************************************/
 ADDRESS LINK "BLGYRXM search,control,srcharg,srchres";
```

This is an example of how to code inputs in REXX for text search transactions.

```
/*********************************************************************/
/* Partial REXX code to illustrate defining control and input        */
/* information for a text search.  Assume that a session has         */
/* already been initialized.                                         */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/* Define control information.                                       */
/*********************************************************************/
data_view_name='BLMPROB'               /* Use problem inq data view */
separator_character=','
/*********************************************************************/
/* Specify control information to use.                               */
/*********************************************************************/
control.1='data_view_name'
control.2='separator_character'         /* defaults to comma if not  */
                                        /* specified                 */
/*********************************************************************/
/* Define input data.  Assume you want to search on all problems     */
/* that have the following words in the description text or          */
/* resolution text:                                                  */
/*    import                                OR                       */
/*    filter                                AND                      */
/*    Microsoft PowerPoint                  AND                      */
/*    Freelance Graphics                                             */
/*                                                                   */
/*********************************************************************/
text.1='(import OR filter)'
text.2='AND "Microsoft PowerPoint"'
text.3='AND "Freelance graphics"'
text.0=3

/*********************************************************************/
/* Explicitly define the search arguments.                           */
/*********************************************************************/
srcharg.1.?name= 'TEXT.'

/*********************************************************************/
/* Call BLGYRXM.  Search outputs will be returned into array with    */
/* stem 'srchres.'.                                                  */
/*********************************************************************/
ADDRESS LINK "BLGYRXM search,control,srcharg,srchres"
```

This is an example of how to code inputs in REXX for retrieve transactions.

```
/*********************************************************************/
/* Partial REXX code to illustrate defining control and input        */
/* information for a qualified retrieve.  Assume a session           */
/* has already been initialized.                                     */
/*********************************************************************/

/*********************************************************************/
/* Define control information                                        */
/*********************************************************************/
pidt_name='BLGYPRR';                  /* use problem retrieve PIDT   */
alias_table='PROBAL';                 /* use alias table PROBAL      */
rnid_symbol='00001200';               /* retrieve record 1200        */
/*********************************************************************/
/* Specify control information to use.                               */
```

```
/*********************************************************************/
control.1='pidt_name';                /* required control info       */
control.2='alias_table';              /* use alias table processing  */
control.3='rnid_symbol';              /* record ID to use            */
control.0=3;
/*********************************************************************/
/* Explicitly identify which data items to retrieve.                 */
/* Note that only the name is required, no data is necessary.        */
/* We want to retrieve the status, severity, date opened,            */
/* assignee name, and reporter department from the record.           */
/*                                                                   */
/* These variable names must be defined in the PROBAL alias          */
/* table with the correct s-words.                                   */
/*********************************************************************/
input.1.?name='STATUS';
input.2.?name='SEVERITY';
input.3.?name='DATE_OPENED';
input.4.?name='ASSIGNEE_NAME';
input.5.?name='REPORTER_DEPT';
input.0=5;
/*********************************************************************/
/* call BLGYRXM.  Return data into array with stem 'output.'.        */
/*********************************************************************/
ADDRESS LINK "BLGYRXM retrieve,control,input,output";
```

Output Data

The HLAPI/REXX interface translates PDBs from the HLAPI into REXX variables. Return codes, error codes, record IDs, retrieved fields, and search match information are just some of the types of output information that the HLAPI/REXX interface returns to your application.

HLAPI/REXX Interface Return Code

The HLAPI/REXX interface sets REXX variable RC with a return code. See "HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, and REXX HLAPI/USS Return Codes" on page 343 for descriptions of all possible return codes.

REXX Variable Data or Access Errors

Sometimes REXX variable data is not correct, or errors occur while the HLAPI/REXX interface is trying to access REXX variables. For these kinds of errors the HLAPI/REXX interface sets variables to provide information about the errors.

When the HLAPI/REXX interface accesses REXX variables, it uses REXX service IRXEXCOM. If access fails, IRXEXCOM returns a nonzero return code and possibly a 1-character hexadecimal code that describes an error accessing a specific variable. The HLAPI/REXX interface sets variable BLG_IRXEXCOM_RC with the access failure return code. Variable BLG_SHVRET is set with the 1-character hexadecimal code. BLG_VARNAME is set to the name of the REXX variable that has the error.

HLAPI Return and Reason Code

The HLAPI/REXX interface provides to your application certain results from the performance of the HLAPI. The value of HLAPI register 15 is set in variable BLG_R15. The HLAPI return code (HICARETC) is set in variable BLG_RC. REXX variable BLG_REAS is set to the value of the HLAPI reason code (HICAREAS). If the HLAPI is not started, these variables are set to null.

For HLAPI return codes 4, 8, and 12, the REXX variable BLG_REAS is set to the decimal conversion of HICAREAS. For example, if HICAREAS contains X'00000010', then BLG_REAS is set to 16.

For HLAPI return code 16 (indicating an abend) BLG_REAS is set to a 1-character code of U, indicating a user abend, or S, indicating a system abend, and 3 hexadecimal characters that identify the abend code. This information comes from HICAREAS. For example, if HICAREAS contains X'000C4000', then BLG_REAS is set to S0C4. If HICAREAS contains X'00000010', then BLG_REAS is set to U010.

HLAPI Timeout Occurrence

If the HLAPI finds a timeout during a transaction, it terminates the HLAPI and LLAPI sessions and returns a return code of 12 and a reason code of 2. You cannot perform any other transactions without performing an initialization to start another API session. Ensure that your REXX program is initializing checks for timeouts (variables RC=200, BLG_RC=12, and BLG_REAS=2) and take appropriate action. Because the HLAPI session has ended, your REXX EXEC has only two choices: stop processing HLAPI/REXX interface transactions completely, or perform an initialization and resume processing transactions. If you attempt any transaction other than initialization after a timeout, the HLAPI/REXX interface sets the variable RC to 20.

Note: Ensure that you set the control field TIMEOUT_INTERVAL to a large enough number at initialization to prevent inadvertent timeouts from occurring. This number varies according to your environment and what work you are trying to accomplish. A setting of 6 minutes (360 seconds) is a reasonable setting to start with. As you become more experienced, you can adjust the setting to your particular situation.

Errors Flagged for Input Items

The HLAPI/REXX interface returns to your REXX program any error flags that the HLAPI sets for input items. It sets *input.n*.?code to the error code. The variable BLG_IERR is set to a string of indexes of the input array that were flagged with errors, each separated by a blank. BLG_IERR has a maximum length of 250 characters and can contain up to 50 indexes of input array elements flagged with errors. If either of these limits is reached, a + follows the end of the BLG_IERR string. The + indicates that one or more input items have *input.n*.?code set and the application should traverse the input array to find them. See PDBCODE on page 222 for error codes.

This example shows how input error codes can be processed.

```
/*********************************************************************/
/* Partial REXX code showing input error processing                  */
/*********************************************************************/

STATUS='ELIMINATED';
SYSTEM_NAME='NEWYORKSYS15';

input.6.?name='STATUS';
input.7.?name='SYSTEM_NAME';

ADDRESS LINK "BLGYRXM update,control,input,outp";
/*********************************************************************/
/* STATUS's value is too long and SYSTEM_NAME's value is too long    */
/* input.6.?code is set to 'L' (data too long)                       */
/* input.7.?code is set to 'L' (data too long)                       */
/* BLG_IERR is set to '6 7' by the REXX API                          */
/*********************************************************************/
```

```
/*********************************************************************/
/* Display errors flagged with the input.                            */
/*********************************************************************/
if blg_ierr/='' then;                 /* any input errors?           */
  do;
    parse var blg_ierr tindex rest;
    do while tindex/='' & tindex/='+'; /*is there another one and    */
                                      /* is it not the end?         */
       say 'Input field' input.tindex.?name 'flagged with error'
                 input.tindex.?code;
       parse var rest tindex rest;
    end;
  end;
```

Information Outputs

Record IDs of records you create, fields you retrieve from a record, and results from searches you conduct are all examples of information outputs.

Each information output item has an associated flag type that gives your REXX application information about the output item. Possible types are:

A The item is a direct-add item.
D The item is a date.
G Specifies that this PDB comes first in a group of one or more related history data items. This is indicated by the associated PIHTSGRP row field set to Y.
H Specifies that this PDB is not the first PDB in a group of several related history data items. This is indicated by the associated PIHTSGRP row field not set to Y.
L The item contains one or more list items with separator characters.
P The item is a phrase item
S The item is a string type of data.
X The item is a text data set identifier.
Blank The output item has no special attributes.

Just as for inputs, you can pass an optional compound variable stem from your application to the HLAPI/REXX interface. This compound variable stem `output` is used as the stem for compound variables that the HLAPI/REXX interface sets to the name, type, error code, and data value associated with each output item. The `output` stem cannot exceed 32 characters. If you do not specify `output`, the HLAPI/REXX interface uses the default `BLG_OUT`.

For nonsearch outputs, the HLAPI/REXX interface sets elements of `output` with the alias name or s-word of the output item (in `output.n.?name`) and the type of this item (in `output.n.?type`). The variable `output.0` is set to the number of output items returned.

When you retrieve freeform text, and you have chosen not to have text returned in data sets, the HLAPI/REXX interface returns the text in an array. `output.n.?name` is set to the array name (not including the period), `output.name.1` through `output.name.n` contain the actual text lines, and `output.name.0` is set to the number of text array elements created.

For outputs from a qualified retrieve transaction, the output items are listed in the same order as the input items you specify for retrieval. The first output item is always the separator character. If an item exists in the retrieve view (that is, is defined in the PIDT that is specified) but has no data, `input.n.?code` is set to `E`. If an item does not exist in the retrieve view, `input.n.?code` is set to `M`. If data exists for the item, the HLAPI/REXX interface returns the data in the same order as you request it. However, the index number of an output item does not necessarily correspond to its index number in the input compound variable.

For outputs from an unqualified retrieve transaction, the order of the output items is the same as the order that is specified in the retrieve PIDT.

For search result outputs, the HLAPI/REXX interface sets the compound variable `output.n.?RNID` with the record ID of the record the search finds. It sets `output.n.?rectype` with the alias name or s-word that identifies the record type of the found record. It sets `output.n.?assoc` with any associated data that the search returns, as long as it is less than 45 characters. It sets `output.?TOTAL` with the total number of matches for the search. And if an error code is generated, `output.n.?code` is set with that error code. If no error occurs, it is set with X'00'.

Examples of Output

The following examples show returns of output items and errors.

This is an example of how to process output items from a retrieve transaction.

```
/*********************************************************************/
/* Partial REXX code showing retrieve transaction output data.       */
/* Assume a session has already been initialized.                    */
/*********************************************************************/

/*********************************************************************/
/* Define control information for the retrieve.                      */
/*********************************************************************/
RNID_SYMBOL='00000305';               /* retrieve record 305         */
PIDT_NAME='PROBRET';                  /* use problem retrieve PIDT   */
ALIAS_TABLE='PROBAL';                 /* use alias table PROBAL      */
TEXT_OPTION='YES';                    /* request text be returned    */
TEXT_MEDIUM='B';                      /* return text in a buffer     */
TEXT_UNITS=100;                       /* maximum of 100 lines of text*/
                                      /* for each type returned      */
TEXT_AREA='B';                        /* want bottom 100 lines       */
TEXT_WIDTH=30;                        /* want 1st 30 characters of   */
                                      /* each line                   */
/*********************************************************************/
/* Specify which control information to use.                         */
/*********************************************************************/
control.1='pidt_name';
control.2='rnid_symbol';
control.3='alias_table';              /* use alias table processing  */
control.4='text_units';
control.5='text_medium';
control.6='text_option';
control.7='text_area';
control.8='text_width';
control.0=8;
/*********************************************************************/
/* Call BLGYRXM.  Data will be returned into array with stem         */
/* 'outputp.'.  Do not specify specific fields to retrieve.          */
/*********************************************************************/
ADDRESS LINK "BLGYRXM retrieve,control,,outputp"
/*********************************************************************/
/* All fields from record 305 that also have a row specification     */
/* in the PIDT are returned by the HLAPI and passed along to the     */
/* REXX program by the HLAPI/REXX. Assume PROBRET PIDT only          */
/* has rows for status, description abstract, serial number, and     */
/* description text and that these fields exist in record 305.       */
/* Assume that PROBAL alias table has these fields defined           */
/* as aliases STATUS, DESCRIPTION, SERIAL_NUMBER, and                */
/* DESCRIPTION_TEXT.  Also assume that PROBAL alias table has these  */
/* fields defined with the correct s-words.                          */
/*                                                                   */
/* The HLAPI/REXX interface sets output variables:                   */
/* OUTPUTP.SEPARATOR_CHARACTER=','                                   */
/* OUTPUTP.STATUS='OPEN'                                             */
/* OUTPUTP.DESCRIPTION='TERMINAL IS BROKEN'                          */
/* OUTPUTP.SERIAL_NUMBER='029977011,ST0023'                          */
/* OUTPUTP.DESCRIPTION_TEXT.1=                                       */
/* 'Bad problem                    91320 08:15:22 APPLID01 MASTER  ' */
/*                                                                   */
/* Retrieve always appends text audit data to the text data          */
/* if audit data is requested                                        */
/* OUTPUTP.DESCRIPTION_TEXT.0=1                                      */
/* OUTPUTP.1.?name='SEPARATOR_CHARACTER'                             */
/* OUTPUTP.1.?type=' '                                               */
/* OUTPUTP.2.?name='STATUS'                                          */
/* OUTPUTP.2.?type=' '                                               */
/* OUTPUTP.3.?name='DESCRIPTION'                                     */
```

```
/* OUTPUTP.3.?type='S'                                              */
/* OUTPUTP.4.?name='SERIAL_NUMBER'                                  */
/* OUTPUTP.4.?type='L'                                              */
/* OUTPUTP.5.?name='DESCRIPTION_TEXT'                               */
/* OUTPUTP.5.?type='X'                                              */
/* OUTPUTP.0=5                                                      */
/* Retrieve always returns an output named SEPARATOR_CHARACTER      */
/* that provides the separator character used for this retrieve.    */
/********************************************************************/
/********************************************************************/
/* Check for timeout and exit - could also do another INIT to restart */
/* the session.  Timeout indicates that the session is terminated.  */
/********************************************************************/
if BLG_RC=12 & BLG_REAS=2 then;        /* timeout occur?            */
  do;
    say 'API timeout occurred - session terminated';
    exit;
  end;
/********************************************************************/
/* Process data returned - display name, type, and value            */
/********************************************************************/
Do i=1 to OUTPUTP.0;
  say '=================== OUTPUT' i ' ===================';
  otype=outputp.i.?type;
  tname=outputp.i.?name;
  if otype=' ' then;
    otype='No special type';
  say tname 'is type '''otype''' and has a value of:';
  select;
    when otype='X' then;
      do j=1 to outputp.tname.0;
        say outputp.tname.j;
      end;
    otherwise;
      say outputp.tname;
  end;
end;
```

This is an example of how to process output items from a qualified retrieve transaction.

```
/********************************************************************/
/* Partial REXX code showing processing of output returned for a    */
/* qualified retrieve (a RETRIEVE where names of items to retrieve  */
/* are specified as inputs).                                        */
/********************************************************************/

/********************************************************************/
/* Define control information for the retrieve.                     */
/********************************************************************/
RNID_SYMBOL='00000305';               /* retrieve record 305        */
PIDT_NAME='BLGYPRR';                  /* use problem retrieve PIDT  */
TEXT_OPTION='YES';                    /* request text be returned   */
TEXT_MEDIUM='B';                      /* return text in a buffer    */
TEXT_UNITS=100;                       /* maximum of 100 lines of text*/
                                      /* for each type returned     */
TEXT_AREA='B';                        /* want bottom 100 lines      */
TEXT_WIDTH=30;                        /* want 1st 30 characters of  */
                                      /* each line                  */
/********************************************************************/
/* Specify which control information to use.                        */
/********************************************************************/
control.1='pidt_name';
control.2='rnid_symbol';
control.4='text_units';
control.5='text_medium';
control.6='text_option';
control.7='text_area';
```

```
control.8='text_width';
control.0=8

/*********************************************************************/
/* Specify which items to retrieve.                                  */
/*********************************************************************/
drop ci.
ci.1.?name='S0B59';
ci.2.?name='S8002';
ci.3.?name='S0BEE';
ci.4.?name='S0B9B';
ci.5.?name='S142F';
ci.6.?name='S0E0F';
ci.7.?name='S0E01';
ci.0=7;



/*********************************************************************/
/* Call BLGYRXM.  Data will be returned into array with stem         */
/* 'outputp.'.  SEPARATOR_CHARACTER is always the first output item.  */
/*********************************************************************/
ADDRESS LINK "BLGYRXM retrieve,control,ci,outputp";

/*********************************************************************/
/* Check for timeout and exit - could also do another INIT to restart */
/* the session.  Timeout indicates that the session is terminated.    */
/*********************************************************************/
if BLG_RC=12 & BLG_REAS=2 then;        /* timeout occur?             */
  do;
    say 'API timeout occurred - session terminated';
    exit;
  end;
/*********************************************************************/
/* Process data returned - display name, type, and value             */
/*********************************************************************/
outs=1;                                /* skip separator char        */
do i=1 to ci.0;
  select;
    when ci.i.?code='E' then;
      say '****' ci.i.?name 'not found in record' rnid_symbol;
    when ci.i.?code='M' then;
      say '****' ci.i.?name 'not defined in retrieve PIDT';
    otherwise;
      do;
        outs=outs+1;
        say '========== OUTPUT' outs 'for input item' i '========';
        otype=outputp.outs.?type;
        tname=outputp.outs.?name;
        if otype=' ' then;
          otype='No special type';
        say tname 'is type '''otype''' and has a value of:';
        select;
          when otype='X' then;
            do j=1 to outputp.tname.0;
              say outputp.tname.j;
            end;
          otherwise
            say outputp.tname
         end;
       end;
  end;
end;
```

Output Messages

The HLAPI/REXX interface generates no error messages on its own. You can indicate at initialization time, however, that you want to chain the HLAPI messages. The HLAPI provides these messages on a PDB chain, and the HLAPI/REXX interface provides them to your REXX program. The HLAPI/REXX interface sets elements of the compound variable `BLG_MSGS` with the data messages that the HLAPI returns. `BLG_MSGS.0` is set with the number of messages returned.

Error Codes

When you provide input information to the APIs for a transaction, error codes can be set in the PIDT for input items. The HLAPI provides the HLAPI/REXX interface with any errors that are flagged in the PIDT. Each error consists of a PIDTSYMB value that identifies the field in error and the associated error code. The HLAPI/REXX interface sets elements of the compound variable `BLG_ERRCODE` with a PIDTSYMB value and error code. `BLG_ERRCODE.`*n*`.?name` receives a PIDTSYMB value, and `BLG_ERRCODE.`*n*`.?code` receives the error code. `BLG_ERRCODE.0` is set to the number of error code items returned. A list of validation codes can be found on page 236.

REXX Reserved Variables

Table 75 describes the reserved REXX variables that the HLAPI/REXX interface uses. The HLAPI/REXX interface defines and sets all of the variables except FREEFORM, FREEFORM.0, and INIT.?PROC.

Table 75. REXX Reserved Variables

Name	Description
BLG_R15	The value of register 15 upon return from the HLAPI. Null if the HLAPI is not called. For specific return codes that set this variable, see page 343.
BLG_RC	The return code from the HLAPI. Null if the HLAPI is not called. “Return Codes” on page 301 contains explanations of return codes.
BLG_REAS	The reason code from the HLAPI. Null if the HLAPI is not called. Explanation of reason codes begins with “Reason Codes for Return Code=0” on page 302.
BLG_VARNAME	Set to the name of the REXX variable that is not valid or that contains data that is not valid. Null if no error causes it to be set.
BLG_IRXEXCOM_RC	Set to the return code from access service IRXEXCOM when a variable access fails. See page 343 for which return code sets this variable.
BLG_SHVRET	Set to a 1-character hexadecimal return code from the access service when an access fails. Additional information on BLG_SHVRET can be found in *REXX/MVS Reference*, SC28–1883.
BLG_IERR	Set to a string of indexes from the input array for which errors are flagged by the HLAPI. Null if no input errors flagged.
FREEFORM	Array that is built by the REXX programmer to define one or more freeform search argument segments.
FREEFORM.0	Variable set by the REXX programmer to indicate how many freeform argument segments are defined.
BLG_OUT	Array providing the name, data, and type for each output item. This variable does not initialize if the call to the HLAPI/REXX interface includes an output stem as a parameter.
BLG_OUT.0	Count of output items returned. Does not initialize if the call to the HLAPI/REXX interface includes an output stem as a parameter.

Table 75. REXX Reserved Variables (continued)

Name	Description
BLG_OUT.?TOTAL	Total number of matches for a search. Does not initialize if the call to the HLAPI/REXX interface includes an output stem as a parameter.
BLG_MSGS	Array providing all messages chained by the HLAPI.
BLG_MSGS.0	Count of messages returned.
BLG_ERRCODE	Array providing PIDT s-word and error code pairs returned by the HLAPI.
BLG_ERRCODE.0	Count of PIDT error codes returned.
BLG_ENVP	HLAPI/REXX environment variable. Set at initialization and dropped at termination.
INIT.?PROC	User sets to T before requesting initialization to enable the HLAPI PDB tracing.

You can find a fully-coded example of a HLAPI/REXX interface in the Tivoli Information Management for z/OS SAMPLIB (MVS data set SBLMSAMP). Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for information on the person to contact for information on high-level qualifiers of data sets at your site. See "Sample HLAPI/REXX Interface" on page 369 for more information on sample programs.

4

HLAPI Extensions

This chapter describes several HLAPI extensions. This chapter also provides information on how similar HLAPI extensions can be written. All HLAPI extensions are invoked using the HLAPI transaction HL14 (for the REXX HLAPI, the USERTSP transaction is used).

BLGTRPND

BLGTRPND is a HLAPI extension that can be used by an application program to reset all the approver data in a change record from Accept or Reject to Pending. The approval status is also changed to Pending.

Control Data

The HLAPI application uses the HL14 transaction or HLAPI/REXX USERTSP transaction to run BLGTRPND. See “Start User TSP or TSX (HL14)” on page 166 for the control PDBs that are used for an HL14 transction.

Input Data

The following input PDBs are used by BLGTRPND.

RNID_SYMBOL

RNID_SYMBOL is required. The record identifier of the change record to be updated is specified in this PDB.

Output Data

On successful completion HICARETC and HICAREAS are set to 0.

Return Codes

Table 76. BLGTRPND reason and return codes

HICARETC	HICAREAS	Description
0	0	Successful completion.
8	4	Logic error. Internal control blocks could not be located. No approvals are changed.
8	900	Required parameter RNID_SYMBOL is missing.
12	165	BLGTRPND had a syntax error. Check for messages to determine the problem.

BLGTSPCH

BLGTSPCH, supplied with Tivoli Information Management for z/OS, is a HLAPI extension that can be used by an application program to know what character to use as a *not sign* or as an *or bar* in data passed to Tivoli Information Management for z/OS for processing. Call BLGTSPCH using the HL14 transaction (HLAPI/REXX USERTSP) to retrieve the characters. On the HL14 transaction, specify BLGTSPCH as the input PDB TSP_NAME. The *not sign* and *or bar* are returned in output PDBs.

Output Data

The output PDBs are as follows:

Table 77. BLGTSPCH output PDBs

PDBNAME	PDBDATL	PDBDATA
NOT_SIGN	1	The character to be used as the *not sign*; the default is ¬ X'5F'
OR_BAR	1	The character to be used as the *or bar*; the default is | X'4F'

Return Codes

BLGTSPCH sets HICARETC and HICAREAS. This table lists these codes. If the HICARETC is 0, output PDBs will contain the *not sign* and *or bar* characters. If the HICARETC is 12, no output PDBs are generated.

Table 78. BLGTSPCH reason and return codes

HICARETC	HICAREAS	Description
0	0	Successful processing. The character to be used for the *not sign* is returned in the PDB NOT_SIGN and the character to be used for the *or bar* is returned in the output PDB OR_BAR
12	165	BLGTSPCH had a syntax error. Check for Tivoli Information Management for z/OS messages to determine the problem.

BLGTXINQ

BLGTXINQ, supplied with Tivoli Information Management for z/OS, is a HLAPI extension that can be used by an application program to perform a search and return multiple data items from the records located by the search. It can be used instead of a Record Retrieve (HL06) and Record Inquiry (HL11) transaction. In addition, the items retrieved can be returned in a sorted order.

Note: List processor data and free-form text data cannot be returned by BLGTXINQ. Like all HLAPI transactions. BLGTXINQ requires the use of control PDBs and input PDBs. BLGTXINQ can be used by all HLAPI clients.

Note: You should modify BLGTXINQ only in the section designated for user modifications.

Control Data

The HLAPI application invokes this transaction by specifying control PDB TRANSACTION_ID with a value of HL14. If you are using the HLAPI/REXX TSP, specify transaction USERTSP). See "Start User TSP or TSX (HL14)" on page 166 for other control PDBs that can be used on an HL14 transaction.

Input Data

The following input PDBs are used by BLGTXINQ.

TSP_NAME

TSP_NAME is required and must be the value BLGTXINQ.

SEARCH_ARGUMENT

The optional input PDB SEARCH_ARGUMENT is used to pass any keywords, s-words, p-words, and search operators needed to locate the desired records. The format of SEARCH_ARGUMENT is similar to the SEARCH command used by an interactive user. Structured searches (s-word and p-word) are supported by sending the s-word or p-word index in the form of !Snnnn for s-words and !Pnnnn for p-words, where nnnn is the hex dictionary index for the s-word or p-word. Search operators and other search keywords are processed exactly as if they were processed using an interactive SEARCH command. The *Tivoli Information Management for z/OS User's Guide* contains additional information on the SEARCH command.

Note: Alias names for s-words and p-words are not supported.

If SEARCH_ARGUMENT is not used, then a search with no keywords is performed. This typically would return the first 32768 records in the database unless the values specified in the SORTPFX keyword of the BLGPARMS macro in the session member is used to limit the number of records.

SEARCH_ARGUMENT examples

- The following will locate only problem records (s-word S0032) that were NOT entered in 1999 (P7E5A is the p-word for 'DATE/') and are CLOSED:

  ```
  !s0032 ¬!P7E5A1999. STAC/CLOSED
  ```

- The following will locate only problem records (s-word S0032) that were entered in 1999 and are CLOSED.

  ```
  !s0032 DATE/1999/01/01 -12/31 STAC/CLOSED
  ```

- The following will locate ANY records that were entered in 1999 and are CLOSED.

  ```
  DATE/1999/01/31 -12/31 STAC/CLOSED
  ```

TABLE_PANEL

TABLE_PANEL is an optional input PDB and is used to determine the data that is returned from the located records and the order of the returned records. The value of TABLE_PANEL is set to an existing Search Results List (SRL) table panel name. The table panel could be one used by your interactive users, or one created for use with BLGTXINQ. Like the SRL that an interactive user sees when viewing the results of an interactive search, the data returned for each records located by SEARCH_ARGUMENT is determined by the columns of the SRL shown on the SRL table panel. Any data shown on the SRL table panel is returned to the application by BLGTXINQ. Data that is not shown on the SRL is not returned. If the table panel used the 'sort on prefix' option, then the located records will be

returned sorted using the prefix value. By using different SRL table panels, it is possible to have different data returned or sorted on different prefixes.

If TABLE_PANEL is not used or is all blanks, then the default table BLGlTSRL is used unless control panel BLG1A120 has been changed to select another default table panel. If TABLE_PANEL contains only an asterisk (*), then any SEARCH_ARGUMENT is ignored, and a list of the available SRL table panel names and their descriptions is returned. See the TABLE command for more information displaying the list of available SRL table panels.

Output Data

On successful completion (HICARETC less than 8), the output PDB TOTAL_HITS is set to the total number of hits the SEARCH_ARGUMENT received. This value could exceed 32 768. This is analogous to the information text “Lines 1 to 32 768 of 108 000 matches” that appears on an SRL when a search produces more than 32 768 matches. Output PDB HITS is set to the actual number of matches returned by this transaction. This value is limited to a maximum of 32 768. Both TOTAL_HITS and HITS are controlled by the values used in the session member of the SORTPFX keyword of the BLGPARMS macro.

If HITS is greater than 0, one or more output PDBs, COLUMN1 through COLUMNn are returned, where n is the value contained in output PDB COLUMNS. There is one data item for each record located in each COLUMNn output PDB. The length of the each data item in a COLUMNn output PDB is the same and is returned in PDBDATW. COLUMNS is the number of data columns on the table panel. The command prefix column on the table panel is ignored and not returned. Output PDB COLUMNS is returned only if HICARETC is less than 8. The s-word index (Snnnn) or P-word index (Pnnnn) value for each data column of a table panel is returned in output PDB COLUMNINDEX1 through COLUMNINDEXn, where n is the value of COLUMNS. The COLUMNINDEXn values allow the application to identify the data returned in each COLUMNn.

Return Codes

TSCARETC and TSCAREAS are set as follows for normal processing. In the event of a syntax error, BLGYAPSR is used to set return 165 if there is a syntax error.

Table 79. BLGTXINQ reason and return codes

TSCARETC Return Code	TSCAREAS Reason Code	Description
0	0	Search_argument received hits. Up to 32768 are returned even if more than 32768 hits were received. ■ *Total_Hits* = The number of hits the search received. ■ *Hits* = The number of hits returned to user. ■ *Columns* = The number of COLUMNn returned to user. ■ *COLUMNn*= Data for that column. **Note:** The following are considered ‘good’ searches: ■ Search_Argument received 0 hits ■ Search returned unsorted (SORTPFX N2 exceeded)
4	901	Too many hits received. SORTPFX N1 exceeded, so the SRL was not shown. *Total_Hits* is set to the number of hits received. *Hits* is set to zero.

Table 79. BLGTXINQ reason and return codes (continued)

TSCARETC Return Code	TSCAREAS Reason Code	Description
4	902	Too many hits received. SRCHLIMIT canceled the search before it completed. *Total_Hits* and *Hits* are both set to zero.
8	901	Table_Panel not found in any panel data set.
8	902	Table_Panel value is not a table panel.
8	903	No valid data columns located on table_panel. At least one non-command column required.
8	904	No valid command column found on table_panel, or more than one command column found. At lease one command column is required.
8	905	Table_Panel value is a table panel but it cannot be processed as a Search Results list.
8	906	Panel BLG1TTBL has been modified and cannot be processed. Make sure that BLG1TTBL is using sequence numbers and that the column information has not been modified.
8	907	Search argument is greater than 512 characters or search argument has over 40 keywords.
12	904	Internal error. SIGNAL_LINE indicates the line number that caused processing to be terminated. Also check the messages that are returned.
12	905	Database error. SIGNAL_LINE indicates the line number that cause processing to be terminated. Check for messages which may be returned and check for messages in SYSPRINT.
12	999	Unexpected error. Check for Tivoli Information Management for z/OS messages.
12	165	TSX syntax error. Check for Tivoli Information Management for z/OS messages.
12	166	This is a general error that you can use when writing TSXs.

Usage Notes

If you use this extension and expect to return large numbers of records, you may need to increase the value that is used for TIMEOUT_INTERVAL.

The width of the table panel used by BLGTXINQ is limited to 80 characters. A table panel can contain scrollable columns. Data that can only be seen by scrolling a column will not be returned. The exception is the last column on the table panel. All data contained in the last column will be returned if NO is specified for “Last column not foldable” (see *Tivoli Information Management for z/OS Panel Modification Facility Guide*) when the table panel was created. Therefore, when creating a table panel for use with BLGTXINQ, the last column can be narrowest (that is, take up the fewest characters) but still return the most data. Consider letting the last column contain the data field with the most data so that the other columns can be as wide as needed to display the desired data. When choosing the width for the last column for performance reasons, make it as wide as possible. The

minimum width for the last column is the maximum width of the data to be displayed in the field divided by 10 and then rounded up to the next whole number.

For example, assume that S0E0F (typically the description field) is to be displayed in the last column and the maximum width for s0E0F on your panels is 45 characters. So, 45/10=4.5; the minimum width for the last column would be 5 (4.5 rounded up). Making the last column wider than 5 characters would improve performance, but is not necessary.

An important difference of BLGTXINQ from an HL11 Inquiry transaction is the BLGTXINQ will not return the RNID unless the RNID value is shown in a table panel column. If RNID is one of the columns shown on the table panel, then the RNID values will be returned in the corresponding COLUMNn output PDB. The output PDB COLUMNINDEXn is also returned, which would allow the application to determine which output COLUMNn PDB contained the RNID values.

Another difference from HL11 is that the width of the values of a returned column could be any length. The ASSOCIATED_DATA returned on an HL11 was fixed and limited to 45 characters.

When TABLE_PANEL is set to an asterisk (*), on return TOTAL_HITS and HITS are set to the number of available table panels. COLUMNS is set to 2; COLUMN1 will contain the table panel names (PDBDATW=8_ and COLUMN2 will contain the table panel descriptions. COLUMNINDEX1 will be set to s1741 (s-word or table panel names) and COLUMNINDEX2 will be set to s1745 (table panel description). You can find additional information on the TABLE command in the *Tivoli Information Management for z/OS User's Guide*.

You should carefully consider the values used for the SORTPFX keyword on the BLGPARMS macro in the session member used by this HLAPI application. SORTPFX controls how many matches (N1 value) can be processed, and how sorting of an SRL will be performed (N2 and N3 values). See the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for more information on SORTPFX.

Writing HLAPI Extensions

A HLAPI extension must be a TSX to be able to use the GETAPIDATA to retrieve input PDBs created by a HLAPI application. The TSX can create output PDBs to be returned to the HLAPI application using SETAPIDATA. In addition, SETAPIDATA allows HICARETC and HICAREAS to be set to allow the TSX to indicate success or failure of the HLAPI extension processing. The TSX to be run as a HLAPI extension is determined by the value of the TSP_NAME input PDB on the HL14 transaction. Like any other TSX, a TSX used as a HLAPI extension must be a member in the BLGTSX data set concatenation. The HLAPI extension TSX can access a TSP by using a TSX LINK control line. Any TSX control line can be used by the HLAPI TSX extension. Most Tivoli Information Management for z/OS commands can be used. Therefore, the processing that your HLAPI extension can do is quite extensive. Observe these rules:

- The API must be active to use GETAPIDATA and SETAPIDATA. Therefore, if your TSX could be executed by an interactive user, be certain to include a test to see if the API is active. See "Usage notes for HLAPI Extensions" on page 271 for additional considerations.

- Before your TSX exits, it should always resume any suspended sessions and return to the primary option panel. This can be done with an ;INITIALIZE command.
- Do not write a long running or never ending HLAPI extension TSX. Your TSX should do what is needed and return. Remember that the application program is waiting for your TSX to complete. If your TSX does not complete before TIMEOUT_INTERVAL expires, the HLAPI will terminate the session and the application using the HLAPI will receive a HICARETC=12 and HICAREAS=2 to indicate that the HLAPI transaction timed out. This indicates a problem in your HLAPI extension TSX.
- Keep your return codes simple. it is recommended that HICARETC=0 means that your TSX extension processed successfully. A HICARETC=4 could mean that processing was successful, but a message may have been issued. A HICARETC of 8 or more should mean that processing was not successful. HICAREAS can be used to provide more detail for a given HICARETC. Most of the documented API return and reason codes do not apply to HL14 transactions.
- Return code 12 reason code 165 is reserved for syntax-type errors in a TSX. Return code 12 reason code 166 is reserved for other, more general errors in a TSX. Your TSX should have a SYNTAX routine to set return code 12 and reason code 165 in the event of a syntax-type error and another routine to set return code 12 and reason code 166 in the event of a general TSX error.
- Like any HLAPI transaction, your TSX will run using the PRIVILEGE_CLASS and APPLICATION_ID that is currently active. Your HL14 can include control PDBs to change the PRIVILEGE_CLASS or APPLICATION_ID.
- Your TSX should test to ensure that the API is active before doing a GETAPIDATA or SETAPIDATA. If the TSX is not running under the HLAPI, those TSX control lines will fail. See the example of using user exit BLGTSAPI to determine if the API is active.

HLAPI REXX Example

```
/* HLAPI REXX example */
trans='USERTSP'
tsp_name = 'BLGTXINQ'
/* Find all Problem records entered in November 98 */
search_argument     = "!S0032 DATE/1998/11/01 -31"
table_panel         = 'BLG1TSRL'
INPUT.1.?NAME = 'tsp_name'
INPUT.2.?NAME = 'search_argument'
INPUT.3.?NAME = 'table_panel'
INPUT.0       = 3

Drop hits
Address LINK "BLGYRXM" trans || ',,INPUT,OUTPUT'
Do i = 1 to OUTPUT.0 while (datatype(OUTPUT.0,'W'))
   OTYPE = OUTPUT.i.?TYPE
   TNAME = OUTPUT.i.?NAME
   If datatype(OUTPUT.TNAME.0,'W') & OUTPUT.i.?TYPE <> 'X' then
      Do J = 0 to OUTPUT.TNAME.0
         Call value TNAME || '.'j ,OUTPUT.TNAME.J
     End
  Else
     Call value tname, output.tname
End
if datatype(hits,'w') then
  Do
    Say 'Hits       =' hits
    Say 'Total_Hits =' total_hits
    If hits > 0 then
      Say 'Columns    =' columns
```

```
    Do i = 1 to hits
      temp = ''
      Do n = 1 to columns
       temp = temp || value('COLUMN'n'.'i) || ' '
      End
      /* Say the 1st 79 bytes so it is easy to read on 3270 screen. */
      Say substr(temp,1,79)
    End
End
```

Getting Input Data

A HLAPI extension TSX does not use PIDTs or Data View records to retrieve data. The TSX can obtain input data two ways. When the TSX is invoked, the data contained in USER_PARAMETER_DATA is passed as a parameter and can be accessed using the REXX keyword instructions ARG or PARSE ARG. This is useful when the amount of input data is limited.

The second method is to use the TSX control line GETAPIDATA. The actual input PDB names are determined when you write the TSX; those input PDB names must be used by the HLAPI application calling your TSX HLAPI extension. You can have any number of input PDBs. They can be single item PDBs or multiple item PDBs. Assume that you decided to use ASSIGNEE as a single item input PDB and RNIDS as a multiple item input PDB. The application using your HLAPI extension would code:

```
/* A REXX user of your HLAPI extension would code */
TSP_NAME = 'MYTSXEXT'  /* whatever you call your TSX */
ASSIGNEE = 'HELPDESK'
RNIDS.0 = 3
RNIDS.1 = '00000001'
RNIDS.2 = '00000002'
RNIDS.3 = '00000003'

INPUT.0 = 3
INPUT.1.?NAME = 'TSP_NAME'
INPUT.2.?NAME = 'ASSIGNEE'
INPUT.3.?NAME = 'RNIDS.'

Address LINK "BLGYRXM" 'USERTSP,,INPUT,OUTPUT'
```

Then in the TSX ’MYTSXEXT’ to retrieve these input PDBs you would code:

```
/* Get the list of RNIDs  */
   Call blgtsx 'getapidata','RNIDS','the_input_rnids.'
   If TSCAFRET = 0 then
      If datatype(the_input_rnids.0,'W') then
         Do i = 1 to the_input_rnids.0
            Say 'RNID' i '=' the_input_rnids.i
         End

/* Get the assignee in a stem even though there is only 1 */
   Call blgtsx 'getapidata','ASSIGNEE','person_name.'
   If TSCAFRET = 0 then
      say 'Assignee name is ' person_name.1
```

Return Data

A HLAPI extension TSX does not use PIDTs or Data View records to return data to the HLAPI application. The TSX can only return data by creating output PDBs using the SETAPIDATA TSX control line. You can return as much data as you would like. You can return single item output PDBs or multiple item output PDBs.

To return a single item output PDB with the PDBNAME of STATUS to the HLAPI application user, the HLAPI TSX extension should contain the code:

```
record_status = 'COMPLETE'
 Call BLGTSX 'SetAPIdata','STATUS',record_status
```

To return a multiple item output PDB with the PDBNAME of LIST, you must use a stem. You must also determine the longest item, because all the items will be padded with blanks to the length, so that PDBDATW can be used by the application receiving the data to correctly use it. For example:

```
longest = 0
Do i = 1 to LIST.0
   x = length(LIST.i)
      if x > longest then longest = x
End
Call BLGTSX 'SetAPIdata','LIST','LIST.',LIST.0,longest
```

Two reserved output PDB names that you can use have special meaning. They are HICARETC and HICAREAS. When you set these using SETAPIDATA, they are not returned on the output PDB chain. Their values are used to set the HICA fields with the same name. Applications using the HLAPI use these fields to determine if a HLAPI transaction was successful. To set HICARETC and HICAREAS, the HLAPI TSX extension would code:

```
 Call BLGTSX 'SetAPIdata','HICARETC',my_return_code
 Call BLGTSX 'SetAPIdata','HICAREAS',my_reason_code
```

Note: The values you set are used only if the values in the HICA are zero. If the HICA values HICARETC and HICAREAS are not zero, an error occurred processing the HL14. Refer to “Return and Reason Codes” on page 301 to determine the meaning.

A general guideline for setting return codes is:

4 warning

8 validation or parameter checking

12 processing error and TSX syntax error

16 severe error

Remember that two reason codes for return code 12 are already defined. Return code 12 reason code 165 is defined for TSX syntax errors. Return code 12 reason code 166 is defined as a general error that you can use when writing TSXs. In order to avoid conflict with other established Tivoli Information Management for z/OS return and reason codes, the range of reason codes in the range 900–999 is reserved for your use and Tivoli use in HLAPI extension TSXs. That is, you can use return code 4, reason code 900 through 999; return code 8, reason code 900 through 999; return code 12, reason code 900 through 999; return code 16, reason code 900 through 999.

Usage notes for HLAPI Extensions

You should test to ensure that the TSX is running under the HLAPI before you attempt to use GETAPIDATA or SETAPIDATA. The following code can be used to determine if the HLAPI is active:

```
/* See if the API is active. */
Call BLGTSX 'USEREXIT','BLGTSAPI'

If TSCAFRET = 0 then
```

```
   api_active = 1
else
  Do
    Say 'Sorry HLAPI is not active.'
    Exit
  End
```

Your TSX should include a syntax routine to handle syntax errors. Here is a sample syntax routine that sets the reason code 165 using API user exit BLGYAPSR. Return code 12 is automatically set if BLGYAPSR is used.

```
/* near the top of your TSX include the line */
Signal on syntax

....
....

/* then at the bottom of your TSX include: */
/* ------------------------------------------------------------ */
 /*********************************************************************/
/* Subroutine to display helpful information in the event of a       */
/* syntax or TSX control line parameter error                        */
/*********************************************************************/
syntax:
  errsigl=sigl                    /* Save failing line number        */
  call issuemsg 'SAY',20200,sigl  /* Show failing line number        */
  say strip(SourceLine(errsigl),'T') /*   and the line source        */
  if symbol('BLG_ERROR.0')='VAR' then /* Control line errors?        */
    do i = 1 to BLG_ERROR.0       /* Loop through the messages       */
      say BLG_ERROR.i             /* Display error message           */
    end
  /*********************************************************************/
  /* If API active then set syntax error codes                         */
  /*********************************************************************/
  if blgapi=1 then
    do
      Call BLGTSX 'SetAPIData','HICARETC',12
      Call BLGTSX 'SetAPIData','HICAREAS',165
    end
  exit 8
```

5

Tips for Writing an API Application

This chapter describes the steps typically involved in creating an application that uses the Tivoli Information Management for z/OS APIs. Every programmer has a certain technique or style for designing applications, so think of this chapter more as a set of guidelines rather than as a set of rules.

Determine What You Want Your Application to Do

The first step in creating an application is to determine exactly what you want it to do. After you decide what you want the application to do, consider:

- Which Tivoli Information Management for z/OS functions (for example, create or update) does it use?
- Which record types (for example, problem or change) does it use?
- Which fields (for example, status or assignee name) does it use?

Determine Which Application ID You Want to Use

Determine the application ID you want your application to use. Create your own or pick one that is already defined, but make sure that the ID you select is in a privilege class that has the authority to perform the functions you want on the record types you want. If your system administrator has chosen to set APISECURITY=ON parameter in the BLX-SP parameters, the MVS user IDs that run your application must be allowed to use the application ID that you choose for your application. For more information about APISECURITY, see “API Security” on page 287.

Determine Which Level of the API You Want to Use

Decide which API to use based on the operating characteristics of each one. “Introduction to the Application Program Interfaces” on page 1 provides information about the APIs that can help you decide. Consider these things when you make your choice:

- If performance is more critical than ease of coding, and your applications will run on MVS, consider the LLAPI. For many types of applications, the performance benefit of using the LLAPI instead of the HLAPI is not significant. The kinds of applications that can provide significantly better performance when using the LLAPI are those that perform many similar inquiry transactions. For example, applications that search many times for problem records.
- If ease of coding is more important than the performance benefit of using the LLAPI, then use the HLAPI.
- If in the future you might be writing applications for one of the remote platforms the HLAPI supports, consider using the HLAPI, because the programming interface it provides is the same across all platforms it supports.

- If you want to write the application in REXX, use the HLAPI/REXX interface. The HLAPI/REXX interface enables you to use the HLAPI with REXX programs. In addition, a REXX programming interface similar to HLAPI/REXX is available on the OS/2 and AIX platforms, so if you plan to use these platforms in the future, your REXX application can be used on either of them.
- If you want to write your application in Java, use the HLAPI for Java. This is only available from the remote HLAPI platforms. Additional information on the HLAPI for Java can be found in the *Tivoli Information Management for z/OS Client Installation and User's Guide*.

Determine Whether You Must Modify LLAPI TSPs

The LLAPI uses TSPs to perform its transactions. The HLAPI indirectly uses the LLAPI TSPs because the HLAPI transactions use the LLAPI. When regular panel processing is used, the LLAPI uses some of your interactive panels as part of its processing for many transactions. When bypass panel processing is used, the LLAPI only uses panels to process the delete transaction. See "API Control Flow" on page 283 for additional information. You might need to modify the LLAPI TSPs for any of the following:

- You have customized Tivoli Information Management for z/OS initialization, record create processing or update processing.
- You want to create or update user-defined record types.
- You want to enable certain API functions

For more information, see the following:

- Page 17 for information on LLAPI TSPs.
- "Tailoring the Application Program Interfaces" on page 289 for information on modifying these TSPs.
- "Terminal Simulator Panels" on page 349 for a description of these TSPs.

Determine Whether You Must Build New API Tables

You can use static PIDTs built by BLGUT8 or PIDTs generated from data view records to define the "view" of data for your application. If you want to use data view records, you must build them along with the associated data attribute and validation records.

If the application you are creating is for working with your own user-defined record types or Integration Facility records, or you want to use views for the data that are different from those that are shipped with Tivoli Information Management for z/OS, you might need new API tables. See "Field Validation Using the Field Validation Module BLGPPFVM" on page 279 for information about the PIDTs, the Table Build Utility, and validation records. Here are some situations that require new static tables:

- If fields have been added, deleted, or changed, and you have not already built new tables for the record type/function pairs you want to use.
- If performance or minimized virtual storage use is critical and you do not already have customized tables for the record type/function pairs you want to use. In this case, customized tables means tables that contain only the set of fields you want to deal with.

Determine Which API Control Block Mapping Macros You Need

The macros identified in this section are provided as programming interfaces for customers by Tivoli Information Management for z/OS.

CAUTION:
Do not use as programming interfaces any Tivoli Information Management for z/OS macros other than those identified in this list.

Determining which API control block mapping macros you need is not required for applications using the HLAPI/REXX interface.

Use these macros or control block definitions to map the storage that is used to communicate with the API. You use these storage definitions to allow your application to send information to the API and to access information that is returned by the API. Macros and control block definitions map the various data areas used by the API and your application.

Review the control block definitions and transaction examples in "Using the LLAPI" on page 15 for the transactions you want to use. Then locate or define API control block mapping macros from the following lists:

- For the LLAPI
  - PICA. Use with all transactions and activities.
  - PIDT. Use with create, update, add record relations, or inquiry transactions.
  - PIPT. Use with create or update transactions and with the IBM®-supplied utility (BLGPPFVM) to validate the data being added to the record.
  - PIAT. Use with inquiry transactions with freeform arguments
  - PIRT. Use with inquiry transactions.
  - PIMB. Use if your application checks messages issued by Tivoli Information Management for z/OS while running your transactions.
  - PIHT. Use if your application requests history data processing on the LLAPI retrieve record (T100) transaction.
- For the HLAPI
  - HICA. Use with all transactions and activities.
  - PDB. Use with all transactions and activities.

Assembler DSECTs and "C" header files are provided for all of these control blocks. They are stored in the ABLMSAMP distribution library, which is created during the installation of Tivoli Information Management for z/OS. If you use another programming language, you must define the control block mappings. Someone in your organization may have already done this. For more information on these mappings, see "LLAPI Structures" on page 100, "HLAPI Structures" on page 216, or study the assembler DSECTs or C header files provided by Tivoli Information Management for z/OS. The Tivoli Information Management for z/OS SAMPLIB (MVS data set SBLMSAMP) contains sample structure definitions as described in Table 80 on page 276.

Table 80. Sample Structure Definitions

Language	API	SAMPLIB Member
C	LLAPI	BLMLLCU
C	HLAPI	BLMHLCU
PL/I	HLAPI	BLMHLPS

See "Sample HLAPI/REXX Interface" on page 369 for more information on sample programs.

Determine If You Want To Use Data Model Records

With earlier releases of Tivoli Information Management for z/OS, information about the format and structure of the data (what was called in "Data Model Records" on page 11 the *composition* of the data records) was stored in panels and in static PIDTs built by BLGUT8. Now, a means is provided of storing this "data model" in records rather than in panels. This eliminates the storage, maintenance, integrity, and security concerns of static PIDTs and PIPTs.

In order to use the data model records which were created previously and saved in the database, you must signal this in either of two methods, depending on whether you are using the LLAPI or HLAPI:

- In the LLAPI, the flag PICADMRC must be set to **Y** in order to use the data model function. This indicates that the PIDT name is a data view record ID and should be used to build the PIDT during the requested transaction.
- In the HLAPI, the PDB DATA_VIEW_NAME must specify a data view name either as an alias or a data view record ID.

Determine If You Want To Bypass Panel Processing

In earlier releases of Tivoli Information Management for z/OS, in order to accommodate locally modified panel flows, you had to modify the API TSPs to follow the modified panel flow.

Now, function is added so that your APIs can bypass panel flow. More information on the means of bypassing panel processing can be found in "API Control Flow" on page 283.

Write Your Application

Using the control block mapping macros you located or defined previously, write your application. It is a good idea to start by coding the initialization and termination transactions and then testing. In general, it is a good practice to code a small piece of the application and test it, then add another small piece and test it, and so on. Using this method facilitates debugging your application because you can more easily isolate the problem area.

Do not link-edit the API server into the same load module as your application because this practice consumes excess storage. Also, if the API server is enhanced in the future, you would have to relink all your API applications. It is more efficient to use either a load and call method or an MVS LINK to access the server. The AMODE and RMODE parameters

you use to link-edit your application can affect which method you choose to access the server. See page 145 for more information on HLAPI transactions, and page 15 for more information on LLAPI transactions.

Include a print statement that prints the return code and reason code from the PICA (LLAPI) or HICA (HLAPI) after each transaction to help you identify any errors detected by the API. When your application works as you want it, you can remove this print statement or modify it to print only unexpected return and reason codes.

If you use the LLAPI, specify either P or B in PICAMSGD and some value (greater than zero) in PICASPLI so that a transaction log, including any Tivoli Information Management for z/OS messages, is produced. A transaction log is helpful in debugging. You may also find it helpful to print out PICA fields at various critical locations in your application, such as immediately before calls to the LLAPI server module BLGYSRVR. Many problems occur because the PICA contains values that are not valid. Verify that each PICA field contains the value that you think you set it to. Use the PICA field definitions explained in "Using the LLAPI" on page 15 as a guide to appropriate values for the PICA fields. Finally, verify that your application does not try to set PICA fields that should only be set by the LLAPI.

If you use the HLAPI, specify either P or B in the HLIMSG_OPTION PDB and some value (greater than zero) in the SPOOL_INTERVAL PDB so that a transaction log, including any Tivoli Information Management for z/OS messages, is produced. You can print information (in HLAPILOG) about each PDB that is sent to the HLAPI or returned from the HLAPI. Change the setting for PDBPROC to T in the PDB that is named TRANSACTION_ID at HLAPI initialization. This can be helpful in debugging because it lets you trace the flow of data to and from the HLAPI.

If you use the HLAPI/REXX interface, it might be useful, if you must diagnose your program, to output all possible error information that is set in your program after each transaction. Consider this only if a transaction returns a non-zero return code. Refer to the Tivoli Information Management for z/OS SAMPLIB (MVS data set SBLMSAMP) for an example of a procedure that outputs error information. Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for information on the person to contact for information on high-level qualifiers of data sets at your site. See "Sample HLAPI/REXX Interface" on page 369 for more information on sample programs.

You might also find it helpful to allocate a SYSPRINT DD statement when you use the APIs so that error information can be written to SYSPRINT.

6

Field Validation Using the Field Validation Module BLGPPFVM

This chapter provides a description of field validation using the Field Validation Module BLGPPFVM.

Using BLGPPFVM To Validate Data Fields

The field validation module BLGPPFVM is an independent load module you can use to validate field data according to patterns specified in a PIPT associated with a PIDT. BLGPPFVM can also be used to convert the case of data according to the settings in PIDTCGMX and PIDTCDCA (described on page on page 130). The HLAPI also uses this module when you choose field validation for HLAPI create or update transactions.

When you are using the LLAPI, your application must load the module into its address space and then call the module for each validation request. The routine validates the specified field and issues the results through return codes.

Note: The LLAPI can call the validation routine, which may return an error code. The two reasons for which the LLAPI will call the validation routine are:

- If you have specified PIDTEQRP=Y and have an = in the first position of the pattern
- If a value has been provided for the PIDTVSWD field

This is the call syntax for the LLAPI field validation module:

```
<label> CALL BLGPPFVM(parameter list)
```

Figure 12 on page 280 shows the parameter list (PLIST) structure, as it appears to an assembler language program, that calls the interface field validation module.

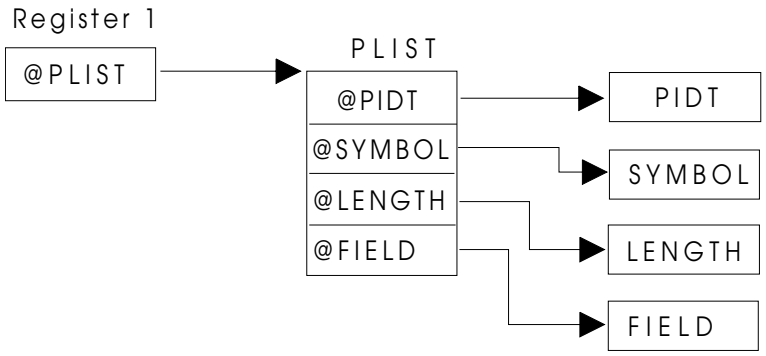

Figure 12. Input Parameter List for BLGPPFVM

When you use the HLAPI, you must set the PDBPROC field of each input chain PDB to V to start the field validation module and cause the PDBDATA field contents of the PDBs to be validated when you process an appropriate transaction.

Input

The required inputs for BLGPPFVM are:

PIDT address The pointer of the PIDT used to validate the field

PIDT symbolic field name The pointer to the symbolic field name of the field to validate

Data field length The pointer to the length of the data field to validate

Data field address The pointer to the data field to validate.

Codes from BLGPPFVM

Return codes indicate the results of the validation. The validation module returns the following return code values.

000 (X'000')

Explanation: Validation was successful for this field.

004 (X'004')

Explanation: Data does not match any validation patterns.

008 (X'008')

Explanation: Field symbolic name was not found in PIDT.

012 (X'00C')

Explanation: PIPT structure is not valid.

016 (X'010')

Explanation: PIDT error.

1. PIDT structure is not valid.
2. A dynamic PIDT was specified. This is not allowed.
3. The PIPT specified cannot be used because it is associated with a dynamic PIDT.

020 (X'014')

Explanation: PIDT contains no entries.

024 (X'018')

Explanation: Field symbolic name was not found in PIPT.

028 (X'01C')

Explanation: PIDT contains a zero value in field PIDTFPAT.

032 (X'020')

Explanation: Length of data to verify is zero.

036 (X'024')

Explanation: Pointer to the data to be verified contains zero.

040 (X'028')

Explanation: An unknown pattern validation character was found.

044 (X'02C')

Explanation: An R or V value is too large in a pattern.

048 (X'030')

Explanation: No R or V value was found in a pattern.

052 (X'034')

Explanation: A literal pattern does not end with a >.

056 (X'038')

Explanation: An R or V value is too small in a pattern.

060 (X'03C')

Explanation: An unknown validation data type was encountered.

064 (X'040')

Explanation: An internal logic error has occurred during the processing of mixed data.

068 (X'044')

Explanation: Field contains mixed data that is not valid.

072 (X'048')

Explanation: An imbedded blank was found in the response. Imbedded blanks are not allowed.

7

API Control Flow

The LLAPI uses TSPs to control the processing of LLAPI transactions. (Because the HLAPI uses the LLAPI to perform its processing, these TSPs are important to the HLAPI as well as the LLAPI).

LLAPI Modes of Operation

The LLAPI has two modes of operation:

Panel processing

The TSPs that control the processing of transactions that file records (create, update, and add record relation) flow through some of your panels. The main router TSP is BLGAPI00. It runs user exits and other TSPs to process LLAPI transactions. If you have modified the Tivoli Information Management for z/OS initialization process you might have to modify TSP BLGAPI00. You need to modify TSPs BLGAPI02, BLGAPI05, and BLGAPI09 if you have modified any of the following selections:

- The selections that start the Tivoli Information Management for z/OS application
- The selections that start record creation
- The selections that file records.

If you write applications that process customized Tivoli Information Management for z/OS records, and you do not choose to bypass panel processing, you might need to perform setup steps or tailor the LLAPI TSPs to correctly process these records. See “Tips for Writing an API Application” on page 273, “Tailoring the Application Program Interfaces” on page 289 and “Terminal Simulator Panels” on page 349 for more information.

The create and update transactions use TSPs BLGAPI02 and BLGAPI05. The add record relations transaction uses TSP BLGAPI09. These TSPs use some of the panels that their corresponding interactive transactions use. For example, TSP BLGAPI02 makes a selection that starts program exit BLG01050.

The LLAPI performs record file processing for create and update transactions by using Selection 9 (File Record) on summary panels. It processes the record just as if you had used the panel interface. That is, certain data fields, such as Date last altered, Time last altered, and Time entered, are automatically set by Tivoli Information Management for z/OS.

Bypass panel processing

Prior to Tivoli Information Management for z/OS Version 6.3, you were dependent on your installation’s customized panels to initialize the API and create and update records via the API. Beginning with Tivoli Information Management for z/OS

Version 6.3, you can "bypass" panel processing. If you choose to bypass panel processing, this must be established at initialization–in T001 for the LLAPI or in HL01 for the HLAPI.

If you are using bypass panel processing, you must use data model records when a record is created or updated. If you choose to bypass panel processing, API transactions are performed by the main router TSP BLGAPIDI. BLGAPIDI uses other TSPs and user exits to perform API processing. TSP BLGAPIPX performs the functions of BLGAPI02, BLGAPI05, and BLGAPI09. BLGAPIPX calls user exits, and does not need to be modified to support your local panel flow.

File processing is performed by user exit BLGYAPRF and a file control panel is not used. You can define in the data view record the name of a TSP to run after create and update file to perform processing such as notification.

In order to use the bypass panel processing and thereby bypass panel flow, you must provide information in either of two methods, depending on whether you are using the LLAPI or the HLAPI:

- In the LLAPI, the flag PICADRIF must be set to Y for the LLAPI to specify bypass panel processing.
- In the HLAPI, the PDB BYPASS_PANEL_PROCESSING must be set to YES to specify bypass panel processing.

In panel processing, the panels obtained some necessary information and provided it to the API. If you choose to bypass panel processing, you must collect certain information in the data view record in order to perform the *create* function. The information that you must collect for a *create* is:

Product s-word
The product s-word must be specified in the data view record. In this case, the product s-word is automatically contained in the generated PIDT. For more information about product s-words and how to create your own product, refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*.

Record access panel
The record access panel must be specified in the data view record. In this case, the record access panel is automatically contained in the generated PIDT. (Both the Product s-word and the Record access panel are found in the record on an *update record* or *add record relation* transaction, so in these cases you do not need to specify this information in the data view record.)

In panel processing, data such as date and time last modified is added directly to the record during file processing. When panel processing is bypassed, these direct add fields must be defined as data attribute records and included within the data view record in the order in which they are to be added to the record. The *Tivoli Information Management for z/OS Panel Modification Facility Guide* contains an example of how to do this.

Depending on whether you choose to bypass panel processing, the TSP flow is different. Based on your implementation, it may be necessary to change these TSPs.

You can use data model records by specifying, for the LLAPI, the PICA field PICADMRC=Y, or for the HLAPI, specify a value for PDB DATA_VIEW_NAME which

specifies a data view name either as an alias or data view RNID. An advantage of this enhancement is that PIDTs and PIPTs are dynamically created in storage using the records. This eliminates the storage, maintenance, and security concerns of PIDTs and PIPTs. This also eliminates the need to use the Table Build Utility (BLGUT8) to build data tables (PIDTs) and validation pattern tables (PIPTs) that the APIs use.

8

API Security

Security checking ensures that a user has the authority to use the value specified in **PICAUSRN**. This security checking is optional and is implemented by the system administrator.

Note: In order to understand the security function, it is necessary to distinguish between two IDs:

— **The MVS user ID used to sign on to MVS**
This can be any of

- A TSO user ID of an interactive user
- A batch job user ID
- A remote application Security ID

— **The application name specified by the application**
This is specified in the PICAUSRN field (see "Low-Level Program Interface Communications Area (PICA)" on page 101) or in the APPLICATION_ID control PDB (see "Parameter Data Definition" on page 225).

Security Implementation

The security function is implemented in the following steps:

- The Tivoli Information Management for z/OS database administrator identifies the application names which will be used to access the Tivoli Information Management for z/OS database through the APIs. These are specified in the PICAUSRN field during LLAPI initialization or the APPLICATION_ID PDB for the HLAPI. These *PICAUSRN application names* are then added to the appropriate privilege classes in the Tivoli Information Management for z/OS database.
- The system administrator uses the RDEFINE RACF command to create a general resource profile for each of these *PICAUSRN application names*. The IBM-supplied class **INFOMAN** is used as the general resource class. This is the same resource class used to control access to the Tivoli Information Management for z/OS data sets. The *MVS user IDs* are then added to the access lists for the appropriate general resource profiles and are given read access authority. Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for additional information on these specifications.
- The Tivoli Information Management for z/OS administrator ensures that the required keyword **APISECURITY** in the BLX-SP startup parameters member is set to **ON**.
- For most transactions, it is possible to change the name of the current application ID by changing the APPLICATION_ID value. During processing, the BLX-SP compares the

MVS user ID with the PICAUSRN provided. If the two values specified are not the same and if **APISECURITY=ON**, a check is made by using the **RACROUTE REQUEST=AUTH** macro to verify read access. If the *MVS user ID* is not in the access list or else if no profile and access list have been defined for the PICAUSRN, an error is returned. The request for a user connection to the BLX-SP fails with an ABEND code 702 and the LLAPI returns a Return Code 12 Reason Code 160 which indicates that the *MVS user ID* is not authorized because of the mismatch with the *PICAUSRN application name*.

9

Tailoring the Application Program Interfaces

PIDTs provide the structure for information on how the APIs define, reference, and list data. You can tailor the data tables to provide different views of the data in a record so that your applications can limit the scope of the data they process. The static data tables are stored in the Report Format Table data set. Your applications can use private PIDT data sets through the use of invocation session members. You can use data model records as an alternative to building static PIDTs.

You can use data view records as an alternate way to define the data composition of records that your application processes. The APIs generate PIDTs directly from the data view records.

You can tailor the APIs in these ways:
- Data table tailoring or data view creation
- User-defined record support
- Terminal simulator panel tailoring.

Prior to Version 6.3, in order to modify panel flow, you had to modify the API TSPs to follow the modified panel flow. In order to modify certain panels, you also had to modify the TSPs.

Now, function is added so that your applications can direct the APIs to bypass panel processing. Additional information about the means to bypass panel processing is contained in “API Control Flow” on page 283.

Tailoring Data Tables

Data table tailoring is the process of specifying the statement criteria used to build a PIDT. The statements can contain all or some of the fields that are associated with a Tivoli Information Management for z/OS record, and that you can collect through existing interactive dialogs.

For example, you want to specify a problem create table named PROBREP that contains only a customized view of the reporter fields because you want to use this table with an application that reports problems. Assume that you want the following fields collected:
- Reporter name
- Reporter department
- Reporter phone
- Reporter location
- Date occurred

- Time occurred
- Status
- Problem type
- Description abstract.

Use the problem create data table statements and copy the FIELD statements specified in the reported data section of the table specifications.

Note: Data table statements used to build all PIDTs and PIPTs are shipped as samples. The name of the member containing problem create data table statements is BLGYPRCS.

Also, copy the TABLE statement (changing the name BLGYPRC to PROBREP) and copy the first FIELD statement that specifies the record type. Finally, specify an ETABLE statement. Now the statements to define the table are complete.

Run the table build utility using PDS panel members that are offloaded from your VSAM panel data sets by using BLGUT6F. The table build utility produces a PIDT named PROBREP and a PIPT named PROBREPP. Refer to the *Tivoli Information Management for z/OS Operation and Maintenance Reference* for more information.

Note: When running the Table Build Utility, use a private table data set and copy the members to the operational data set later.

You can use data model records as an alternative to building static PIDTs.

User-Defined Record Support

To implement support of user-defined models of record data using static PIDTs:

1. Use utility BLGUT6F to copy all of the panels defined for the record type from a Tivoli Information Management for z/OS VSAM panel data set to a partitioned data set.
2. Define the transactions you want to use against the record.
3. Define the tables you want to provide to the API for each transaction you want to use. Remember that each transaction type requires its own data tables.
4. Code the table specification statements for each required table.
5. Run BLGUT8 to build the tables.
6. Update the record processing control panels. See “Record Process Panels” on page 361 for more information.
7. Modify transaction TSPs if necessary. See “Terminal Simulator Panels” on page 349 for descriptions of the TSPs used by the APIs.
8. For complex models of data, you may need to create additional add record relation tables.

Specify Tivoli Information Management for z/OS records using unique record type s-words. You must add these s-word specifications to the record processing control panels BLG1AACP and BLG1AAUP if you are using static PIDTs and panel processing. See “Record Process Panels” on page 361 for more information on these control panels. You must also specify the name of the associated record summary panel in the control panels. When the APIs run record create and update transactions, the LLAPI loads these panels so that the transaction TSPs can simulate record file responses.

Panel processing uses some of the panels in your interactive panel flow. If you model the user record summary panels after the shipped summary panels, little or no tailoring of the appropriate TSPs is required. The APIs perform record file processing by using selection 9 on summary panels. If user record summary panels do not use selection 9, you must change the panel or the transaction TSP. You can copy the user record summary panel to a new panel and change file selection to 9 so the APIs can use that panel and file records. Panels BLG1AACP and BLG1AAUP tell the API which summary panel the file selection for the record resides on.

To use the panel specified in panel BLG1AAUP as the summary panel, specify an authorization code of `0001` for that panel in BLG1AAUP. If you are using bypass panel processing, you must use data model records if you are using transactions that file records (create, update, or add record relation). You do not have to modify the API TSPs that control bypass panel processing to support your customized panels and you do not have to modify panels BLG1AACP and BLG1AAUP.

When to Tailor Terminal Simulator Panels

Note: The information in this section applies if you are using panel processing.

LLAPI TSPs may need tailoring because of unique installation requirements. You can tailor these TSPs if the overall logic flow does not affect the operation of the API subtask. Your modifications must work for all record types that your application uses because all create, update, and add record relation transactions use these TSPs.

The TSPs, as shipped, return control to the controller module BLGYAPCP specified on TSP panel BLGAPI00. See “Terminal Simulator Panels” on page 349 for a discussion of API TSPs.

Listed below are some situations that require tailoring of the LLAPI TSPs. These changes must be made if you use either the HLAPI or the LLAPI.

- The IRC `INIT,3,2` does not access the management application.

  TSP BLGAPI00 requires modification, and other TSPs require modification depending on which record functions you use in your application.

- The IRC `INIT,3,2` does not display the panel to make a selection to start program exit BLG01050; or the panel is displayed, but selection 5 does not start the program exit.

  TSP BLGAPI02 requires modification.

- Selection 9 from the record summary panel does not file the record when you attempt to create, update, or add record relations.

  TSPs BLGAPI02, BLGAPI05 and BLGAPI09 require modification, depending on which transactions your application performs.

10

LLAPI User Exits

Most LLAPI user exits are designed for use only within the LLAPI environment. If any exit, other than BLGEXDEL and BLGTSAPI, detects that it is operating outside this environment, the exit forces Tivoli Information Management for z/OS to end abnormally with return code 700, reason code 32. User exits BLGTSAPI and BLGEXDEL are exceptions. They can operate outside the LLAPI environment.

BLGEXDEL - Delete Unusable Record

CAUTION:
This user exit deletes the root VSAM key without checking for authority. To protect your database, have the TSP that calls this user exit check for authorization.

User exit BLGEXDEL deletes the record using the root VSAM key passed in the TSCA variable data area. The root VSAM key is in character format (0-9, A-F). BLGEXDEL does not uncognize information contained in the record by updating the SDIDS. You must run the SDIDS build utility BLGUT1 after running this user exit to uncognize the record. Until BLGUT1 is run, the record may show as deleted on a search results list.

If this user exit is used in the API, the database number is taken from PICADBID. If this user exit is used outside the API, the database number is taken from the user's profile. If the profile does not contain a database number, database 5 is used.

Table 81 lists the return and reason codes that are returned by the exit.

Table 81. BLGEXDEL reason and return codes

TSCAFRET Return Code	TSCAFRES Reason Code	Description
0	0	Successful completion. Record deleted.
4	4	Variable data length (TSACVDAL) is not equal to 8.
4	8	Database not found.
4	12	Database is not read/write.
4	16	Record enqueue on root VSAM key failed.
8	4	Storage allocation error.
16	4	Internal control blocks not found.
0	4	Storage allocation error. Some records were deleted.
0	8	Database access error. Some records were deleted.
8	8	Storage allocation error.

Table 81. BLGEXDEL reason and return codes (continued)

TSCAFRET Return Code	TSCAFRES Reason Code	Description
8	12	Record not found.
8	16	Cannot delete root VSAM key X'00000000'.

BLGJAUTH - Check Authorization

User exit BLGJAUTH checks whether the user's privilege class has the required authorization for the user to perform the requested action.

Input to BLGJAUTH is a 4-byte authorization code passed in the TSCA variable data area by the calling TSP.

Table 82 lists the return and reason codes that the exit returns.

Table 82. BLGJAUTH reason and return codes

TSCAFRET Return Code	TSCAFRES Reason Code	Description
0	0	Successful completion. The user is authorized to perform the requested action.
4	0	The user is not authorized to perform the requested action.
8	0	The length of the variable data is not 4.
16	0	Tivoli Information Management for z/OS internal control blocks cannot be found.

BLGYAPCP - LLAPI Control Processor

User exit BLGYAPCP, with TSP BLGAPI00 or TSP BLGAPIDI, performs the routing and control processing for the LLAPI API subtask.

BLGYAPCP processes many transactions within code segments. When a TSP implements a transaction, BLGYAPCP copies the transaction code stored in PICA field PICATRAN left justified to TSCA field TSCAUFLD. BLGYAPCP then returns to TSP BLGAPI00 or TSP BLGAPIDI to complete transaction processing by linking to a transaction TSP. BLGAPI00 (or BLGAPIDI) performs the link by testing the transaction code passed in TSCAUFLD.

BLGYAPGP - Retrieve Panel Name

User exit BLGYAPGP retrieves the name of a summary panel for use in record create or update processing. BLGYAPGP also verifies the authority of the application to perform the entry transaction requested. BLGYAPGP also checks fields PIDTUSEF, PICATXAU, and PICAHIST to determine whether dynamic PIDT, text audit data, or history data processing was requested. The exit then sets a return and reason code in the TSCA to indicate which functions were requested. Create (T102), update (T105), and add record relation (T109) transactions use this exit.

If you are using panel processing, summary panel names are stored in the LLAPI record processing control panels BLG1AACP and BLG1AAUP. The value stored in PIDT field

PIDTUSEF determines which of these control panels to use. Each control line of these panels specifies an s-word and a target summary panel name. The record type s-word specified in the PIDT used to perform the transaction becomes the scan search argument. The exit compares this search argument with each control line s-word in the control panel until a match occurs. BLGYAPGP extracts the target panel name and stores it in the TSCA variable data area and sets the variable data area length TSCAVDAL.

Table 83 lists the return and reason codes that are returned by the exit. If the value in TSCAFRES is greater than zero, the value PICDABLE (74) is stored in PIVREAS. This disables the history data, text audit data, and dynamic PIDT processing if the new versions of BLGAPI02 and BLGAPI05 are not put into the panels data set. History data and text audit data are also disabled in BLGAPIPX.

Table 83. BLGYAPGP reason and return codes

TSCAFRET Return Code	TSCAFRES Reason Code	Description
0	0	Successful completion. PIDTUSEF¬=D, PICATXAU¬=Y, PICAHIST¬=Y.
4	0	A summary panel s-word cannot be located, or the application is not authorized to perform the function.
4	1	Successful completion. PIDTUSEF¬=D, PICATXAU¬=Y, PICAHIST=Y.
4	2	Successful completion. PIDTUSEF¬=D, PICATXAU=Y, PICAHIST¬=Y.
4	3	Successful completion. PIDTUSEF¬=D, PICATXAU=Y, PICAHIST=Y.
4	4	Successful completion. PIDTUSEF=D, PICATXAU¬=Y, PICAHIST¬=Y.
4	5	Successful completion. PIDTUSEF=D, PICATXAU¬=Y, PICAHIST=Y.
4	6	Successful completion. PIDTUSEF=D, PICATXAU=Y, PICAHIST¬=Y.
4	7	Successful completion. PIDTUSEF=D, PICATXAU=Y, PICAHIST=Y.

BLGYAPBR - Record Build Processor

User exit BLGYAPBR retrieves data stored in the LLAPI data structures, converts it to Tivoli Information Management for z/OS internal record form, and adds it to the database record. Create (T102), update (T105), and add record relation (T109) transactions use this exit.

The exit sets TSCA return code field TSCAFRET to 4 when the exit finds any processing errors.

BLGYAPSR - Set LLAPI Reason Code

User exit BLGYAPSR retrieves data from the TSCA variable data area and converts it to a reason code that it passes back in PICA field PICAREAS in the LLAPI. If the length of the data in the variable data area is anything other than 4, BLGYAPSR takes the first 2 characters, converts them to numerics, and stores the result in PICAREAS. For example, with length=5, data=98765, PICAREAS is set to 98. With length=3, data=987, PICAREAS is still set to 98.

If the length of the data in the variable data area is 4, and if all 4 characters are EBCDIC numbers, BLGYAPSR converts all the data to numeric and places the result in PICAREAS. Length=4, data=9876 results in PICAREAS being set to 9876.

The exit sets TSCA return code field TSCAFRET to 0 when the exit completes.

You can use user exit BLGYAPSR to set a reason code in the PICAREAS field when the TSP returns. You must use reason codes 1000 to 9999 for user definition. If BLGYAPSR sets a reason code, the associated return code is 12. Instead of using BLGYAPSR, the user TSP can return status in the parameter area passed in PICAPARM.

BLGYAPBU - Retrieve Record ID

User exit BLGYAPBU retrieves the record ID or root VSAM key specified in PICA field PICARNID. If a record ID is retrieved, it is appended to the TSCA variable data area. If a root VSAM key is retrieved, it is converted to a record ID and the record ID is appended to the TSCA variable data area. If the conversion from a root VSAM key to a record ID fails, the root VSAM key is appended to the TSCA variable data area. Update (T105), add record relation (T109), and delete (T110) transactions use this exit. Table 84 lists the return and reason codes that are returned by the exit.

Table 84. BLGYAPBU reason and return codes

TSCAFRET Return Code	TSCAFRES Reason Code	Description
0	0	Successful completion. The record ID is placed in the TSCA variable data area.
4	0	TSCA variable data area overflow. The data is not added to the TSCA variable data area.
4	8	The root VSAM key is place in the TSCA variable data area. An error occurred displaying the record.
4	16	The root VSAM key is placed in the TSCA variable data area, but the root VSAM key is not valid.

BLGYAPUP - Verify Record Update

User exit BLGYAPUP verifies that the record specified to the API is updated. Update (T105) and add record relation (T109) transactions use this exit.

The exit sets TSCA return code field TSCAFRET to 4 if the record is not being updated.

BLGRESET- Reset all Approvals to Pending

User exit BLGYAPUP changes all the approval status data for change records from Approval Provided and Approval Rejected to Approval Pending.

Table 85. BLGRESET Return and Reason Codes

Return Code (TSCAFRET)	Reason Code (TSCAFRES)	Description
0	0	Successful completion.
8	4	Logic error. Internal control blocks could not be located. No approvals are changed.

BLGTSAPI - Test for LLAPI Environment

User exit BLGTSAPI determines if the LLAPI environment is active. The exit sets TSCA return code field TCSAFRET to 0 when the LLAPI environment is active and to 4 when it is not active. There is no input to the exit. The output is TSCA return code field TSCAFRET.

BLGYAPIS - Set Product

User exit BLGYAPIS is called by TSP BLGAPIPX when the API is active. For a create, BLGYAPIS gets the product from the data view record with the product s-word and visible phrase. It then gets the record access panel from the data view record with the record access panel s-word. For an update, BLGYAPIS reads the record from the database and sets the product stored in the record. It then checks record authorization and searches the record for an owning or a transfer-to class. If found, BLGYAPIS then verifies the authority of the application to perform the entry transaction requested.

Table 86 lists the return and reason codes that are returned by the exit.

Table 86. BLGYAPIS reason and return codes

TSCAFRET Return Code	TSCAFRES Reason Code	Description
0	0	No errors.
4	0	A record error occurred. The API reason code is set.
8	0	No product s-word or visible phrase was found on a create transaction. The product s-word must be specified in a data view record.
10	0	On an update or add record relations transaction, an owning or transfer-to class was found. The user does not have authorization to update the record. The API reason code is set.

Table 86. BLGYAPIS reason and return codes (continued)

TSCAFRET Return Code	TSCAFRES Reason Code	Description
12	0	A record access panel error occurred. Either no record access panel was found on a create transaction, the record access panel found is not a control panel, or it could not be loaded. The record access panel must be specified in a data view record. The API reason code is set.
16	0	Invalid program exit call. The transaction must either be a create, update, or add record relations.

BLGYAPRF - File Record

User exit BLGYAPRF is used to file a record that has been processed by user exit BLGYAPBR.

If no error occurs, then program exit BLG01214 is called and the record is filed. If an API error occurs, the record is dequeued, and BLGYAPRF ends.

Table 87 lists the return and reason codes that are returned by the exit.

Table 87. BLGYAPRF reason and return codes

TSCAFRET Return Code	TSCAFRES Reason Code	Description
0	0	The record filed successfully.
4	0	An error occurred while filing the record. The API reason code was set.

A

Record Type and Function PIDT Tables

Table 88 specifies which static PIDT table is defined for use with API transactions. These static PIDTs are shipped with the Tivoli Information Management for z/OS licensed program. The PIDTs shipped in SBLMFMT are for using the standard 10–character date panels that are shipped with Tivoli Information Management for z/OS. The PIDTs which include data fields are also shipped in SBLMFMT.

PIDT to Record SERVICE Transaction Cross-Reference

Table 88. PIDT to Record SERVICE Transaction Matrix

Record Type Description	Record Index	Retrieve T100/HL06	Create T102/HL08	Update T105/HL09	Inquiry T107/HL11
Problem record	S0032	BLGYPRR	BLGYPRC	BLGYPRU	BLGYPRI
Change record	S0B06	BLGYCHR	BLGYCHC	BLGYCHU	BLGYCHI
Activity record	S0B07	BLGYACR	BLGYACC	BLGYACU	BLGYACI
Center record	S0B0C	BLGYDCR	BLGYDCC	BLGYDCU	BLGYDCI
System record	S0B0E	BLGYSYR	BLGYSYC	BLGYSYU	BLGYSYI
Hardware component	S0B0F	BLGYHCR	BLGYHCC	BLGYHCU	BLGYHCI
Hardware subcomponent	S0AF8	BLGYHSR	BLGYHSC	BLGYHSU	BLGYHSI
Hardware feature	S0B10	BLGYHFR	BLGYHFC	BLGYHFU	BLGYHFI
Hardware connection	S0B1E	BLGYHXR	BLGYHXC	BLGYHXU	BLGYHXI
Software component	S0B13	BLGYSCR	BLGYSCC	BLGYSCU	BLGYSCI
Software feature	S0B14	BLGYSFR	BLGYSFC	BLGYSFU	BLGYSFI
Software connection	S0B1F	BLGYSXR	BLGYSXC	BLGYSXU	BLGYSXI
Hardware financial	S0B1B	BLGYHNR	BLGYHNC	BLGYHNU	BLGYHNI
Software financial	S0B1A	BLGYSNR	BLGYSNC	BLGYSNU	BLGYSNI
Service record	S0B19	BLGYSVR	BLGYSVC	BLGYSVU	BLGYSVI

PIDT to Record LIST Transaction Cross-Reference

Table 89 specifies which PIDT table is defined for use with a particular record LIST transaction.

Table 89. PIDT to Record LIST Transaction Matrix

Record List Description	Record Index	Inquiry T107/HL11
List change activities	S0B07	BLGYACL
List hardware features	S0B10	BLGYHFL
List hardware connections	S0B1E	BLGYHXL
List software features	S0B14	BLGYSFL
List software connections	S0B1F	BLGYSXL

PIDT to Record ADD Transaction Cross-Reference

Table 90 specifies which PIDT table is defined for use with a particular record Add Record Relation transaction.

Table 90. PIDT to Record ADD Transaction Matrix

Record Description	Record Index	Add T109/HL12
Add change activities	S0B06	BLGYCHA
Add hardware feature	S0B10	BLGYHFA
Add hardware connection	S0B1E	BLGYHXA
Add software feature	S0B14	BLGYSFA
Add software connection	S0B1F	BLGYSXA

B

Return and Reason Codes

This appendix lists the return and reason codes for the HLAPI, LLAPI, HLAPI remote environment servers, and HLAPI clients. Most return and reason codes are returned through the HICARETC and HICAREAS fields located in the HICA. The codes are divided into sections based on the return code. Within each return code section, reason codes are listed in decimal sequence, with the hexadecimal equivalent appearing to the right of the decimal value. Each reason code is accompanied by an explanation and an origin. The origin indicates which part or parts of the API return the reason code. The following terms are used to identify the origin of a reason code:

Client Code returned by a HLAPI client

HLAPI Code returned by the HLAPI

LLAPI Code returned by the LLAPI

Server Code returned by a remote environment server.

Remember, the HLAPI uses the LLAPI, so even though an error might appear to be coming from the HLAPI, it may very well be the LLAPI that produced the error.

This appendix also lists the return codes for the HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, and REXX HLAPI/USS. Because these return codes do not include a reason code, they are listed in a separate section, "HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, and REXX HLAPI/USS Return Codes" on page 343.

Messages that the HLAPI/USS requester option writes in the blmprobe.log on the OS/390 UNIX System Service host where the requester is running can be found in the *Tivoli Information Management for z/OS Messages and Codes*.

Messages that the HLAPI/UNIX requester option writes in the idbprobe.log on the OS/390 UNIX host where the requester is running can be found in the *Tivoli Information Management for z/OS Messages and Codes*.

Note: The range of reason codes in the range 900–999 is reserved for customer use and can be used in coding user-written TSXs.

Return Codes

The High- and Low-Level APIs and the clients and servers return the return codes listed in this section.

000 (X'000')

Explanation: Indicates that the transaction completed successfully. See “Reason Codes for Return Code=0” for the reason codes returned for this return code.

004 (X'004')

Explanation: Indicates that the transaction completed successfully but stored an informational reason code. See “Reason Codes for Return Code=4” for the reason codes returned for this return code.

008 (X'008')

Explanation: Indicates that the transaction was not used and found a syntax or parameter error. See “Reason Codes for Return Code=8” on page 304 for the reason codes returned for this return code.

012 (X'00C')

Explanation: Indicates that the transaction was used but timed out, or an error occurred after the transaction processing successfully completed. See “Reason Codes for Return Code=12” on page 314 for the reason codes returned for this return code.

016 (X'010')

Explanation: Indicates that the transaction was used but did not complete because of an API subtask ABEND. The ABEND code is returned as the reason code in the form of xxsssuuu where sss is the system ABEND code, uuu is a user ABEND code, and xx are do-not-care positions. The user-written application must reinitialize the API environment before issuing any subsequent transactions. See “Reason Codes for Return Code=16” on page 337 for the reason codes returned for this return code.

020 (X'014')

Explanation: Indicates that an error has occurred that caused all active conversations between a Tivoli Information Management for z/OS server and a client to end. See “Reason Codes for Return Code=20” on page 342 for the reason codes returned for this return code.

Reason Codes for Return Code=0

The following reason codes are returned for return code 0 (X'000').

000 (X'000')

Explanation: Successful completion.

Module: HLAPI, LLAPI

004 (X'004')

Explanation: Check transaction T010 issued but processing transaction not complete.

Module: LLAPI

Reason Codes for Return Code=4

The following reason codes are returned for return code 4 (X'004').

001 (X'001')

Explanation: Indicates the LLAPI is already stopped.

Module: LLAPI

002 (X'002')

Explanation: Indicates the LLAPI is already initialized.

Module: LLAPI

003 (X'003')

Explanation: A LLAPI check or sync transaction was issued, and no transaction was pending.

Module: LLAPI

004 (X'004')

Explanation: A LLAPI T006 or T005 transaction was attempted, but the PIDT address in the PICA is zero.

Module: LLAPI

005 (X'005')

Explanation: A LLAPI T005 transaction was attempted, but the PIPT address in the PIDT is zero.

Module: LLAPI

006 (X'006')

Explanation: A LLAPI T007 transaction was attempted, but the PIRT address in the PICA is zero and the search ID in the PICA is zero.

Module: LLAPI

007 (X'007')

Explanation: Create resources have already been allocated.

Module: LLAPI

008 (X'008')

Explanation: Update resources have already been allocated.

Module: LLAPI

009 (X'009')

Explanation: Inquiry resources have already been allocated.

Module: LLAPI

010 (X'00A')

Explanation: Add record relation resources have already been allocated.

Module: LLAPI

011 (X'00B')

Explanation: The PIPT has already been allocated.

Module: LLAPI

012 (X'00C')

Explanation: The application already has the record checked out.

Module: LLAPI

013 (X'00D')

Explanation: The application does not have the record checked out, or another application has the record checked out.

Module: LLAPI

014 (X'00E')

Explanation: A HLAPI HL06 transaction was attempted, but the field HICAINPP is nonzero, and the field TEXT_MEDIUM is set to D. No text data is returned with the record.

Module: HLAPI

071 (X'047')

Explanation: Not all matches for the inquiry are returned for one of the following reasons:

- The count of matches received from the inquiry is greater than the maximum number of matches specified in the SORT PFX-N1 parameter of the BLGPARMS macro for the session-parameters member being used.
- The number of matches requested is greater than the total number of matches received from the inquiry.

Only the matches following the beginning match number are returned in the search results list.

Module: LLAPI

Reason Codes for Return Code=8

A return code of 8 identifies a validation error reason code. Validation error reason codes are reported in two forms: encoded and explicit. Encoded validation reason codes identify one or more errors, and explicit reason codes identify single errors.

Encoded Validation Error Reason Codes

Validation reason codes with values less than 65 536 are encoded to permit the return of multiple validation errors. Encoded validation errors are returned in the low-order 2 bytes (the third and fourth bytes) of PICAREAS. Each bit set on in these low-order 2 bytes of the reason code field represents a validation error that was detected. For example, if you have multiple validation errors of 1, 8, and 64 then a 1-bit would be present in the units, 8s, and 64s positions of the 2 low-order bytes of PICAREAS. This pattern (B'0000000001001001') indicates the errors detected.

The following list contains the possible encoded reason code values returned as the result of error detection during transaction validation. The code line of each encoded validation reason code includes the decimal value of the bit that is set to 1, followed by the binary value, followed by the hexadecimal value.

00001 (B'0000000000000001') (X'0001')

Explanation: PIDT entries are not valid. One or more entries in the PIDT failed validation tests and are flagged with entry error indicators in PIDTCODE. If the HLAPI returns this reason code, an error PDB is returned for each PDB item flagged. HICAERRP points to the first error code PDB in the error chain. Each PDBDATA field contains the PIDTSYMB field contents. Refer to 123 for a list of PIDT codes.

Module: HLAPI, LLAPI

00002 (B'0000000000000010') (X'0002')

Explanation: PICACLSN is not valid. Data stored in PICACLSN is not blank or contains symbols that do not conform to privilege class record naming conventions.

Module: LLAPI

00004 (B'0000000000000100') (X'0004')

Explanation: The field PICAVSAM was set to Y to signal that a VSAM key was being passed in the PICARNID field for either a create record transaction or an add record relations request. Only record IDs are valid for these transactions

Module: LLAPI

00008 (B'0000000000001000') (X'0008')

Explanation: PICARNID required. Data stored in PICARNID is blank or contains symbols that do not conform to database record identifiers.

Module: LLAPI

00016 (B'0000000000010000') (X'0010')

Explanation: PICATABN required. Data stored in PICATABN is blank or contains symbols that do not conform to the operating system naming conventions for partitioned data set members.

Module: LLAPI

00032 (B'0000000000100000') (X'0020')

Explanation: Reserved.

00064 (B'0000000001000000') (X'0040')

Explanation: PICAPIDT not valid

1. The pointer in PICAPIDT is zero, but the transaction requires a PIDT available for use.
2. The pointer in PICAPIDT points to a PIDT different from the one named in PICATABN.
3. The usage defined for the type of PIDT (PIDTUSEF) is not valid for the requested transaction.
4. The PIDT is defined as dynamic (PIDTUSEF= D) and was specified for a transaction that does not allow dynamic PIDTs.
5. The PIDT is marked corrupted due to a previous error processing data model records. The PIDT should be freed, the error corrected, and the PIDT should be requested again.

Module: LLAPI

00128 (B'0000000010000000') (X'0080')

Explanation: PIDTBUFP not valid.

1. The pointer in PIDTBUFP is zero, but the transaction requires a response buffer available for use.
2. The pointer in PIDTBUFP points to a response buffer different from the one allocated for the PIDT.

Module: LLAPI

00256 (B'0000000100000000') (X'0100')

Explanation: PICAREQR not valid. The value in PICAREQR is less than zero. The value of PICAREQR must be zero if the application does not require a PIAT for record inquiry, or it must be positive (number of PIAT rows to allocate) if the application requires a PIAT for use during record inquiry.

Module: LLAPI

00512 (B'0000001000000000') (X'0200')

Explanation: PICAREQL not valid. The value in PICAREQL is not greater than zero. The value of PICAREQL is used to allocate a response buffer and must be large enough to satisfy the application requirements for record create, update, or inquiry transactions. The value in PICAREQL is ignored and not used during a retrieve transaction. If you are using the HLAPI, this can mean that no inputs were specified.

Module: LLAPI

01024 (B'0000010000000000') (X'0400')

Explanation: PIDTPIAT not valid. The pointer in PIDTPIAT points to a PIAT different from the one allocated for the PIDT.

Module: LLAPI

02048 (B'0000100000000000') (X'0800')

Explanation: PIDTPIPT not valid.

1. The name in PIDTPTNM does not match the name of the PIPT that was obtained for the current PIDT.
2. The pointer in PIDTPIPT points to a PIPT different from the one allocated for the PIDT.

Module: LLAPI

04096 (B'0001000000000000') (X'1000')

Explanation: PICASRCH not valid.

1. Index stored in PICASRCH is not blank, and is not in valid s-word index format.
2. Index stored in PICASRCH cannot be found in the PIDT pointed to by PICAPIDT.

Module: LLAPI

08192 (B'0010000000000000') (X'2000')

Explanation: Reserved

16384 (B'0100000000000000') (X'4000')

Explanation: PICAPIRT not valid. The pointer in PIDTPIRT does not point to a PIRT.

Module: LLAPI

32768 (B'1000000000000000') (X'8000')

Explanation: Reserved.

Explicit Validation Error Reason Codes

The following list contains the possible explicit reason code values returned as the result of error detection during transaction validation.

65536 (X'00010000')

Explanation: Incorrect LLAPI transaction code value (PICATRAN). The value specified is not known as a valid transaction code.

Module: LLAPI

65537 (X'00010001')

Explanation: Incorrect application name (PICAUSRN). Either the first character is not alphabetic or a national character (@, #, or $), or the value contains characters other than alphabetic, numeric, or national characters (@, #, or $).

Module: LLAPI

65538 (X'00010002')

Explanation: Incorrect session name (PICASESS). The first character is not alphabetic or the value contains characters other than alphabetic, numeric, or national characters (@, #, or $). The naming convention for session parameter members is BLGSES*aa*, where you supply the suffix *aa* as two alphanumeric characters. If you use a single numeric digit, it is right-justified with a leading zero. If you use a single alphabetic or national character, it is left-justified with a trailing blank.

Module: LLAPI

65539 (X'00010003')

Explanation: Incorrect invocation privilege class name (PICACLSN). The first character is not alphabetic or the value contains characters other than alphabetic, numeric, or special characters (&, /, @, #, or $).

Module: LLAPI

65540 (X'00010004')

Explanation: Error loading the specified session member (PICASESS). The session member cannot be found in any load library using the current concatenation order.

Module: LLAPI

65541 (X'00010005')

Explanation: Previous transaction has not finished. The currently requested transaction is not a T002, T009, or a T010 transaction, and the previous transaction has not yet finished.

Module: LLAPI

65542 (X'00010006')

Explanation: Initialization transaction required. The application has not specified a T001 transaction when PICAENVP is zero.

Module: LLAPI

65543 (X'00010007')

Explanation: API subtask attach failure. The server attempted to attach the API subtask and received a nonzero return code.

Module: LLAPI

65544 (X'00010008')

Explanation: API subtask detach failure. The server attempted to detach the API subtask and received a nonzero return code.

Module: LLAPI

65545 (X'00010009')

Explanation: Storage not available for LLAPI control blocks. The server attempted to allocate storage that could not be obtained.

Module: LLAPI

65546 (X'0001000A')

Explanation: Reserved.

65547(X'0001000B')

Explanation: Reserved.

65548 (X'0001000C')

Explanation: The application has passed an incorrect PICA structure (PICAENVP).

Module: LLAPI

65549 (X'0001000D')

Explanation: Error loading the date validation module. The server attempted to load the date validation routine defined in the session member and received a nonzero return code.

Module: LLAPI

65550 (X'0001000E')

Explanation: Error deleting loaded module. Either the session member or the date validation routine could not be deleted.

Module: LLAPI

65551 (X'0001000F')

Explanation: A required input PDP is missing.

Module: Client

65552 (X'00010010')

Explanation: HICAENVP does not point to a valid HLAPI environment block. If HICAOUTP, HICAMSGP, or HICAERRP point to PDBs created by a previous transaction, the HLAPI cannot free them.

Module: HLAPI

65553 (X'00010011')

Explanation: PDB structure specified is not valid. PDBACRO in a structure passed as a PDB does not contain the string PDB (left-justified and right-padded with a blank), or PDBNAME is blank, or if used on the input chain, PDBDATL is zero.

Module: HLAPI

65554 (X'00010012')

Explanation: Required CONTROL PDBs have not been specified or the address in HICACTLP is zero.

Module: HLAPI

65555 (X'00010013')

Explanation: HLAPI TRANSACTION_ID PDB value is not valid. The transaction ID specified is not supported by the HLAPI.

Module: HLAPI

65556 (X'00010014')

Explanation: PRIVILEGE_CLASS PDB value is not valid. The first character is not alphabetic or the value contains characters other than a-Z, 0-9, &, $, /, #, or @.

Module: HLAPI

65557 (X'00010015')

Explanation: PIDT_NAME PDB value is not valid. The first character is not alphabetic or the value contains characters other than A-Z, 0-9, &, $, #, or @.

Module: HLAPI

65558 (X'00010016')

Explanation: RNID_SYMBOL PDB value is not valid. All characters must be numeric or the first character must be alphabetic and the remaining characters must be A-Z, 0-9, #, $, @, &, or /.

Module: HLAPI

65559 (X'00010017')

Explanation: ALIAS_TABLE PDB value is not valid. First character must be alphabetic and the remaining characters must be A-Z, 0-9, #, or $ with no imbedded blanks.

Module: HLAPI

65560 (X'00010018')

Explanation: The transaction was not successful for one of the following reasons:

- TEXT_DDNAME PDB value is not valid. Characters must be A-Z, 0-9, #, or $ with no imbedded blanks.
- No input PDBs were specified. At least one TEXT_DDNAME PDB must be specified.
- An input PDB other than a TEXT_DDNAME PDB was specified.

Module: HLAPI

65561 (X'00010019')

Explanation: ASSOCIATED_DATA PDB value is not valid. When no alias table processing is specified, the first characters must be S or P and the remaining 4 characters must be hexadecimal.

Module: HLAPI

65562 (X'0001001A')

Explanation: APPLICATION_ID PDB value is not valid. The first character must be either alphabetic or a national character (@, #, $), and the remaining characters must be alphabetic, numeric, or national characters (@, #, $).

Module: HLAPI

65563 (X'0001001B')

Explanation: SESSION_MEMBER PDB value is not valid. First character must be alphabetic and the remaining characters must be alphabetic, numeric, or the national characters # or &; The naming convention for session parameter members is BLGSES*aa* where you supply the suffix *aa* as two alphanumeric characters. If you use a single numeric digit, it is right-justified with a leading zero. If you use a single alphabetic or national character, it is left-justified with a trailing blank.

Module: HLAPI

65564 (X'0001001C')

Explanation: You requested alias table processing but no alias table storage was allocated at session initialization time.

Module: HLAPI

65565 (X'0001001D')

Explanation: Required SEPARATOR_CHARACTER PDB missing. The HLAPI could not locate a SEPARATOR_CHARACTER control PDB.

Module: HLAPI

65566 (X'0001001E')

Explanation: HLAPI previously initialized and an HL01 transaction has been attempted.

Module: HLAPI

65567 (X'0001001F')

Explanation: HLAPI has not been previously initialized. No transaction or a transaction other than HL01 has been requested and no HICAENVP is stored in the HICA. If HICAOUTP, HICAMSGP, or HICAERRP point to PDBs created by a previous transaction, the HLAPI cannot free them.

Module: HLAPI

65568 (X'00010020')

Explanation: TABLE_COUNT PDB value is too large. The maximum number of tables in storage is 256.

Module: HLAPI

65569 (X'00010021')

Explanation: The date specified with the DELETE_HISTORY control PDB could not be converted by the currently enabled date conversion routine.

Module: HLAPI

65570 (X'00010022')

Explanation: The DELETE_HISTORY control PDB was specified but no history data has been retrieved and saved.

Module: HLAPI

65571 (X'00010023')

Explanation: The specified file was not found.

Module: Client

65572 (X'00010024')

Explanation: The specified code page for the client is not supported.

Module: Client

65573 (X'00010025')

Explanation: The specified code page for the server is not supported.

Module: Client

65574 (X'00010026')

Explanation: The control PDB named DATABASE_PROFILE was not found.

Module: Client

65575 (X'00010027')

Explanation: If this is a HLAPI/CICS client application, the control PDB named CICS_USER_ID was not found, or the length of the CICS_USER_ID entered was greater than 8 characters. This PDB is required on HLAPI/CICS client HL01 transactions.

If this is a HLAPI/2, HLAPI/UNIX, HLAPI/NT, or HLAPI/USS client application, the control PDB named SECURITY_ID was not found or the length entered was greater than 8 characters. This PDB is required on HLAPI/2, HLAPI/UNIX, and HLAPI/NT client HL01 transactions.

Module: Client

65576 (X'00010028')

Explanation: The control PDB named PASSWORD was not found.

Module: Client

65577 (X'00010029')

Explanation: Profile processing. Unknown keyword found.

Module: Client

65578 (X'0001002A')

Explanation: Profile processing. Bad parameter.

Module: Client

65579 (X'0001002B')

Explanation: Profile processing. Duplicate keyword. Keyword already specified.

Module: Client

65580 (X'0001002C')

Explanation: Profile processing. A keyword that is required was not specified.

Module: Client

65581 (X'0001002D')

Explanation: Profile processing. ″=″ not specified.

Module: Client

65582 (X'0001002E')

Explanation: Profile processing. Keyword numeric value is out of valid range.

Module: Client

65583 (X'0001002F')

Explanation: Profile processing. The input line is longer than 512 characters.

Module: Client

65584 (X'00010030')

Explanation: Profile processing. Log filename is too long. 260 characters is the maximum.

Module: Client

65585 (X'00010031')

Explanation: Profile processing. The symbolic destination name is longer than 8 characters.

Module: Client

65586 (X'00010032')

Explanation: Client code and requester code are not at the same version level.

Module: Client

65587 (X'00010033')

Explanation: The requested transaction is in progress.

Module: Client

65588 (X'00010034')

Explanation: No transaction is available.

Module: Client

65589 (X'00010035')

Explanation: The control PDB name CICS_PARTNER_ID was not found, or the length of the CICS_PARTNER_ID entered was greater than 8 characters. This PDB is required for HL01 transactions submitted by an HLAPI/CICS client application.

Module: Client

65590 (X'00010036')

Explanation: The control PDB name CICS_CM_TIME_OUT_VALUE was not found, or the length of the CICS_CM_TIME_OUT_VALUE entered was greater than 6 characters or the time specified was not valid. This PDB is required for HL01 transactions submitted by an HLAPI/CICS client application.

Module: Client

65591 (X'00010037')

Explanation: The control PDB name CICS_INTER_TIME_OUT_VALUE was not found, or the length of the CICS_INTER_TIME_OUT_VALUE entered was greater than 6 characters or the time specified was not valid. This PDB is required for HL01 transactions submitted by an HLAPI/CICS client application.

Module: Client

65592 (X'00010038')

Explanation: Neither IDBSYMDESTNAME nor IDBSERVERHOST was found. You must specify one.

Module: Client

65593 (X'00010039')

Explanation: Both IDBSYMDESTNAME and IDBSERVERHOST were found. You can specify only one of them.

Module: Client

65594 (X'0001003A')

Explanation: The name specified for the requester's host is too long.

Module: Client

65595 (X'0001003B')

Explanation: The name specified for IDBSERVERHOST is too long.

Module: Client

65664 (X'00010080')

Explanation: An unsupported date format was specified on the DATE_FORMAT PDB. The transaction was not performed and the date format was set to the default value (database format).

Module: HLAPI

65792 (X'00010100')

Explanation: An error occurred when attempting to load the validation routine BLGPPFVM.

Module: LLAPI

65920 (X'00010180')

Explanation: On an HL06 (retrieve) transaction, you set control PDB TEXT_STREAM equal to YES but either you did not set control PDB TEXT_OPTION equal to YES or you did not set control PDB TEXT_AUDIT_OPTION equal to NO. Set TEXT_OPTION=YES, set TEXT_AUDIT_OPTION=NO, and retry the HL06.

Module: HLAPI

Reason Codes for Return Code=12

The following reason codes apply when the return code is 12 (X'00C').

001 (X'001')

Explanation: Indicates that the application ID was not authorized to complete the transaction.

Module: LLAPI

002 (X'002')

Explanation: API subtask timeout value has been exceeded. A sync transaction can be attempted to continue the current transaction or a terminate transaction can be attempted to stop the LLAPI. If this error was received from the HLAPI, the API session is terminated.

Module: HLAPI, LLAPI

003 (X'003')

Explanation: Indicates that Tivoli Information Management for z/OS detected an error and might have issued a message that can indicate the error detail. The message text is available in the message buffer chain when the application is receiving messages or printed in the activity log when messages are being logged.

Module: LLAPI

004 (X'004')

Explanation: The LLAPI detected a storage allocation error. The LLAPI attempted to allocate work storage and none was available.

Module: LLAPI

005 (X'005')

Explanation: The LLAPI detected a PIMB allocation error. The LLAPI attempted to allocate storage for a PIMB and none was available.

Module: LLAPI

006 (X'006')

Explanation: LLAPI record processing control panel s-word error. Possible causes are:

- The PIDT does not contain a record type s-word.
- The record type s-word is not found in the record processing control panels BLG1AACP and BLG1AAUP.
- The product s-word was not found in the primary entry control panel. The primary entry control panel is specified by the BLGPARMS PANEL keyword. The default primary entry control panel is BLG0ENTR.

Module: LLAPI

007 (X'007')

Explanation: A LLAPI TSP variable data area error occurred (variable data area filled up).

Module: LLAPI

008 (X'008')

Explanation: One or more database access errors have been detected. Additional messages might be on the message chain.

Module: LLAPI

009 (X'009')

Explanation: A text build error has been detected.

Module: LLAPI

010 (X'00A')

Explanation: Record not found.

Module: LLAPI

011 (X'00B')

Explanation: Record is currently being updated by another user or application.

Module: LLAPI

012 (X'00C')

Explanation: Record was busy.

Module: LLAPI

013 (X'00D')

Explanation: Table data set name not specified in the session member.

Module: LLAPI

014 (X'00E')

Explanation: Table data set allocation error.

Module: LLAPI

015 (X'00F')

Explanation: Table data set open error.

Module: LLAPI

016 (X'010')

Explanation: Table data set close error.

Module: LLAPI

017 (X'011')

Explanation: Table data set logical record length is not 80.

Module: LLAPI

018 (X'012')

Explanation: Table data set record format is not fixed.

Module: LLAPI

019 (X'013')

Explanation: Table data set I/O error occurred.

Module: LLAPI

020 (X'014')

Explanation: Table data set member not found.

Module: LLAPI

021 (X'015')

Explanation: Table data set member is not a valid LLAPI table because of one of the following reasons:

- The table member does not contain the correct acronym to identify it as the type of table requested.
- The usage field for the type of PIDT (PIDTUSEF) is not valid for the requested transaction, and the application is not doing dynamic processing with a retrieve transaction (T100).
- The Report Format Table (RFT) data set cannot be found. Either an RFT data set that contains the PIDT being utilized must be allocated to the session running the API or else at least one of the data sets allocated to the RFTDD statement must contain the required PIDT.

Module: LLAPI

022 (X'016')

Explanation: Table data set member is damaged. An end-of-file exit has been taken by the table member read routine. The member might be truncated.

Module: LLAPI

023 (X'017')

Explanation: Maximum record ID value assignment reached.

Module: LLAPI

024 (X'018')

Explanation: PIDT table name specified for transaction is already defined for use by the requested transaction.

Module: LLAPI

025 (X'019')

Explanation: Text data set or buffer processing error detected. Possible errors include data set I/O error or required storage unavailable. In many cases, other messages have been issued. PIDT entry has been flagged. When the HLAPI is used, it extracts the PIDTCODE code and creates an error PDB.

Module: LLAPI

026 (X'01A')

Explanation: Data response was too long; PIDT entry has been flagged. When the HLAPI is used, it extracts the PIDTCODE code and creates an error PDB. Refer to 123 for a list of PIDT codes.

Module: LLAPI

027 (X'01B')

Explanation: LLAPI record processing control panel not found. Possible causes are:

- Either BLG1AACP (create) or BLG1AAUP (update), or both, cannot be found on any read panel data sets specified by the session member used for initialization.
- The primary control panel was not found. This panel cannot be found in any read panel data set specified by the session-parameters member used for initialization. The primary entry control panel is specified by the BLGPARMS PANEL keyword. The default control panel is BLG0ENTR.

Module: LLAPI

028 (X'01C')

Explanation: Create panel selection error. Possible causes are:

- Program exit BLG01050 was never started in the create process (it causes flow to go to the panel named in BLG1AACP for the current record type — the panel name is obtained by user exit BLGYAPGP).
- An error occurred selecting entry (selection 5) from the main options menu.
- The panel actually flowed to when invoking program exit BLG01050 (current panel) does not match the panel specified in the create processing control panel BLG1AACP for this record type.

Module: LLAPI

029 (X'01D')

Explanation: Update panel selection error. The current update summary panel (normal panel flowed to when entering the update command for the record being updated) does not match the panel name specified in the update processing control panel BLG1AAUP for this record type. Possible reasons for this error are:

- Panel BLG1AAUP contains incorrect information on the record type being updated.
- It is not the correct record and is a different type from the record type specified in the PIDT.
- The wrong PIDT is being used.

Module: LLAPI

030 (X'01E')

Explanation: External record ID specified for create is not valid. Possible causes are:

- The name duplicated a record name already in the database.
- The RNID contains symbols not valid for an RNID.
- A system-assigned format name was used for a system-assigned record ID that has not been system-assigned and is not a record ID that was retrieved using a dynamic PIDT.
- A database access error occurred while determining the validity of the record ID (see the accompanying messages).

- A record ID was specified during update.
- The record ID began with a numeric but is not all numeric.

Module: LLAPI

---

031 (X'01F')

Explanation: PIDT entry error. The PIDT entry in error has been flagged as indicated by a nonzero PIDTCODE. When the HLAPI is used, it extracts this PIDTCODE code and creates an error PDB. Refer to 123 for a list of PIDT codes set by the LLAPI (these are in the range of 00 to 46) and to 236 for a list of PIDT codes set by the HLAPI (these are in the range of 50 to 72).

Module: LLAPI

---

032 (X'020')

Explanation: Text data set access error; PIDT entry has been flagged. When the HLAPI is used, it extracts this PIDTCODE code and creates an error PDB. Refer to 123 for a list of PIDT codes.

Module: LLAPI

---

033 (X'021')

Explanation: Error occurred during Tivoli Information Management for z/OS initialization.

Module: LLAPI

---

034 (X'022')

Explanation: Reserved.

---

035 (X'023')

Explanation: Duplicate record IDs exist in the database.

Module: LLAPI

---

036 (X'024')

Explanation: Reserved.

---

037 (X'025')

Explanation: Add Record Relation panel selection error. The current Add Record Relation (update) summary panel (normal panel flowed to when entering the update command for the record being updated) does not match the panel name specified in the update processing control panel BLG1AAUP for this record type. Possible reasons for this error are:

- Panel BLG1AAUP contains incorrect information on the record type being updated.
- It is not the correct record and is a different type from the record type specified in the PIDT.
- The wrong PIDT is being used.

Module: LLAPI

---

038 (X'026')

Explanation: Record access attempted and failed because record is checked out to another application.

Module: LLAPI

039 (X'027')

Explanation: Unexpected quit command was run. A quit command was run by some process other than a T002 transaction. Check the HLAPILOG, APIPRINT, SYSOUT, NETVIEW LOG, and SYSPRINT data sets for additional messages.

Module: LLAPI

040 (X'028')

Explanation: Privilege class record not found, or application ID not found as an eligible user in the privilege class.

Module: LLAPI

041 (X'29')

Explanation: Privilege class record not available.

Module: LLAPI

042 (X'2A')

Explanation: Privilege class record is busy.

Module: LLAPI

043 (X'2B')

Explanation: Incorrect database ID specified.

Module: LLAPI

044 (X'02C')

Explanation: A text data set reallocation error occurred. The HLAPI found an error when it attempted to reallocate one of the text data sets the LLAPI allocates with a default or user-provided DDNAME. The HLAPI stores an error code in the related input PDB when a text data set processing error occurs. This can also occur if the number of data sets is greater than 99.

Module: HLAPI

045 (X'02D')

Explanation: A text data set free error occurred. The HLAPI found an error when it attempted to free one of the text data sets specified on the input chain pointed to by HICAINPP. The PDBCODE field contains the character I.

Module: HLAPI

046 (X'02E')

Explanation: A text data set delete error occurred. The HLAPI found an error when it attempted to delete one of the text data sets specified on the input chain pointed to by HICAINPP. The PDBCODE field contains the character I.

Module: HLAPI

047 (X'02F')

Explanation: The HLAPI requested storage from the system but none was available.

Module: HLAPI

048 (X'030')

Explanation: The HLAPI requested PDB storage from the system but none was available.

Module: HLAPI

049 (X'031')

Explanation: A BLX environment initialization internal error has occurred.

Module: HLAPI

050 (X'032')

Explanation: A BLX environment termination internal error has occurred.

Module: HLAPI

051 (X'033')

Explanation: The HLAPI attempted to delete the validation module BLGPPFVM and found an error.

Module: HLAPI

052 (X'034')

Explanation: The HLAPI attempted to delete the LLAPI server module BLGYSRVR and found an error.

Module: HLAPI

053 (X'035')

Explanation: Reserved.

054 (X'036')

Explanation: Reserved.

055 (X'037')

Explanation: PALT/PIDT symbol not found. The PDBNAME specified cannot be found in either a PALT (if alias processing) or in the specified PIDT. If alias processing, this error could also be an internal symbol that could not be found in the PIDT. Field PDBCODE is set to M.

Module: HLAPI

056 (X'038')

Explanation: A response length error occurred. The value specified in PDBDATL is greater than that specified for an item in a PIDT. This can also occur when the aggregate length of a freeform inquiry argument is greater than 17 bytes, or when default data obtained from the alias table would be stored beyond the length of the response buffer. Field PDBCODE is set to L.

Module: HLAPI

057 (X'039')

Explanation: A response validation error occurred. The value of data specified in PDBDATA does not match validation criteria for the field. Field PDBCODE is set to V. When the HLAPI is used, it extracts this PIDTCODE and creates an error PDB. Refer to 236 for a list of PIDT codes set by the HLAPI (these are in the range of 50 to 72).

Module: HLAPI

058 (X'03A')

Explanation: A resource cleanup error occurred. The HLAPI found an error while freeing transaction resources it had obtained.

Module: HLAPI

059 (X'03B')

Explanation: A response separator error occurred. A list or multiple response is missing a separator character or the separator character specified could not be found in the item.

Module: HLAPI

060 (X'03C')

Explanation: If control PDB TEXT_STREAM is YES, PDBDATW does not equal PDBDATL. If control PDB TEXT_STREAM is not YES or is not specified, one of the following problems occurred: PDB field PDBDATL divided by PDBDATW produced a remainder, or PDBDATW is greater than 132, or the text data set name is longer than 44 characters, or buffer storage and data set processing was mixed for text data.

Module: HLAPI

061 (X'03D')

Explanation: A delete timer exit routine error occurred. The HLAPI attempted to delete the timer exit routine BLGYHLTE.

Module: HLAPI

062 (X'03E')

Explanation: A HLAPI log allocation error occurred after the transaction successfully ended.

Module: HLAPI

063 (X'03F')

Explanation: A HLAPI log write error occurred after the transaction successfully ended.

Module: HLAPI

064 (X'040')

Explanation: Upon HLAPI termination, a HLAPI log release error occurred after an elapsed spool time interval. The transaction successfully ended.

Module: HLAPI

065 (X'041')

Explanation: When this reason code is returned from a HL15 or HL16 transaction, it indicates that a PDB name other than TEXT_DDNAME was specified. Any valid TEXT_DDNAME PDBs may have been processed successfully. The PDB code field will be set with a value of V.

Module: HLAPI

066 (X'042')

Explanation: While attempting to delete history data, no dates were found. Ensure that the following conditions are true:

- The history dates can be converted from external to internal format by the currently enabled date conversion routine.
- The prefix word for date data begins with the characters DAT.
- The date history data was specified with journal first when created.

Module: HLAPI

070 (X'46')

Explanation: Incorrect record count detected on record read.

Module: LLAPI

072 (X'48')

Explanation: The record s-word in the specified PIDT cannot be found. This can be caused by specifying the wrong PIDT.

Module: LLAPI

073 (X'49')

Explanation: During a create (T102), update (T105), or add record relations (T109) transaction, a list item was found with an index value greater than the maximum allowed value of 19 274.

Module: LLAPI

074 (X'4A')

Explanation: One of the following processing options was requested, but the options were not enabled. You can enable them for create (T102) or update (T105) by modifying either TSP BLGAPI02 (create) or BLGAPI05 (update) or BLGAPIPX for create and update if you are using bypass panel processing.

- Use of a dynamic PIDT (PIDTUSEF=D).
- History data processing (PICAHIST=Y).
- Text audit data processing (PICATXAU=Y).

Module: LLAPI

075 (X'4B')

Explanation: TEXTAUD=YES is specified in the BLGPARMS macro, and the application has attempted to delete or replace existing text. Refer to the *Tivoli Information Management for z/OS Program Administration Guide and Reference* for information on the BLGPARMS macro.

Module: LLAPI

076 (X'4C')

Explanation: Text audit data specification error. Audit data was specified for a text line (PICATXAU=Y) and either the date or time did not pass validation, or all audit data fields for a line are empty. Date audit data must be in the form YYDDD where YY is the last 2 digits of the year, and DDD is the Julian date. Time audit data must be in the form HH:MM:SS. The PIDT entry was flagged by the PIDTCODE. Refer to 123 for a list of PIDT codes.

Module: LLAPI

077 (X'4D')

Explanation: Freeform text was specified using the data set method (PICATXTP=D), and the application indicated it was specifying audit data with the text (PICATXAU =Y). The LRECL of a text data set was not 168. The PIDT entry was flagged by the PIDTCODE. Refer to 123 for a list of PIDT codes.

Module: LLAPI

078 (X'4E')

Explanation: No entry in the record was found specifying the record type. The s-word for the record must be found in the create control panel, BLG1AACP.

Module: LLAPI

079 (X'4F')

Explanation: During an update transaction (T105) the s-word, p-word, or panel name specified in PIDTVLDD was not found in the record entry corresponding to the PIDT entry. The PIDT entry has been flagged by the PIDTCODE. Refer to 123 for a list of PIDT codes.

Module: LLAPI

080 (X'50')

Explanation: The PIDT specified has not been migrated to the current format. The PIDT was not loaded.

Module: LLAPI

082 (X'52')

Explanation: During a create transaction (T102), an update transaction (T105), or a delete transaction (T110), internal control structures required for transaction processing were not found.

Module: LLAPI

083 (X'53')

Explanation: During a delete transaction (T110), a LLAPI TSP variable data error occurred (variable data length is not equal to 8).

Module: LLAPI

084 (X'54')

Explanation: During a delete transaction (T110), the enqueue on root VSAM key failed. A possible cause is another user currently holding the enqueue.

Module: LLAPI

085 (X'55')

Explanation: The record in the database was changed after the data was retrieved. The record cannot be updated.

Module: LLAPI

086 (X'56')

Explanation: A history data specification error occurred. The PIHT provided for a record update transaction was not obtained with a record retrieve transaction for the same record ID.

Module: LLAPI

087 (X'57')

Explanation: A history data specification error occurred. The row count (PIHTNUMR) value must not be changed.

Module: LLAPI

088 (X'58')

Explanation: The pointer in PIDTPIHT is not zero and does not point to a valid PIHT.

Module: LLAPI

089 (X'59')

Explanation: PIHT row error. The PIHT row in error was flagged as indicated by a nonzero PIHTCODE. A listing of the PIHT codes can be found in Table 36 on page 132.

Module: LLAPI

090 (X'5A')

Explanation: During an update (T105) transaction, the timestamp last altered s-word was not found in the record.

Module: LLAPI

091 (X'5B')

Explanation: During a record processing transaction, the RNID prefix was not found in the record.

Module: LLAPI

092 (X'5C')

Explanation: One of the following processing options was requested for a record create (T102), update (T105) or delete (T110) transaction, but the API does not have the required authorization as specified in TSP BLGAPI02, BLGAPI05, or BLGAPI10. If you are using bypass panel processing, this authorization for create and update processing options was not specified in BLGAPIPX.

If you are using the Archiver, the privilege class running the Archiver does not have database administration authority.

- Use of a dynamic PIDT (PIDTUSEF=D)
- History data processing (PICAHIST=Y)
- Text audit data processing (PICATXAU=Y)
- Delete a damaged record.

Module: LLAPI

094 (X'5E')

Explanation: During a retrieve transaction (T100) requesting a dynamic PIDT, an attempt was made to retrieve an SRC record.

Module: LLAPI

096 (X'60')

Explanation: A dynamic PIDT processing error occurred. The RNID of the record to be updated does not match the RNID of the record that was retrieved with this dynamic PIDT. This is not valid.

Module: LLAPI

097 (X'61')

Explanation: A dynamic PIDT specification error occurred. The s-word data length (PIDTSWDL) is larger than 10, or the p-word data length is larger than 6. The PIDT entry was flagged by the PIDTCODE. Refer to 123 for a list of PIDT codes.

Module: LLAPI

098 (X'62')

Explanation: A dynamic PIDT build error occurred. Refer to the *Tivoli Information Management for z/OS Diagnosis Guide* for information on contacting IBM.

Module: LLAPI

099 (X'063')

Explanation: The specified search ID already exists. The search was not performed.

Module: LLAPI

100 (X'064')

Explanation: The specified search ID does not exist.

Module: LLAPI

101 (X'00000065')

Explanation: The client system is out of memory.

Module: Client

102 (X'00000066')

Explanation: An error occurred while allocating memory.

Module: Client

103 (X'00000067')

Explanation: The client code set is not compatible with the server code page. Code points cannot be converted between the two.

Module: Client

104 (X'00000068')

Explanation: The client thread has prematurely ended.

Module: Client

105 (X'00000069')

Explanation: The specified conversation security information is not valid.

Module: Client

106 (X'0000006A')

Explanation: A supporting system thread could not be created. No more threads are available.

Module: Client

107 (X'0000006B')

Explanation: A system interrupt has occurred.

Module: Client

108 (X'0000006C')

Explanation: No more pipe resources are available.

Module: Client

109 (X'0000006D')

Explanation: The requester is not available. A requester must be started before HLAPI requester services can be accessed by a client application.

Module: Client

110 (X'0000006E')

Explanation: No more handles are available. Increase handle count.

Module: Client

111 (X'0000006F')

Explanation: Log file failed to open.

Module: Client

112 (X'00000070')

Explanation: An error occurred while archiving the log file.

Module: Client

113 (X'00000071')

Explanation: An error occurred while retrieving an error message.

Module: Client

114 (X'00000072')

Explanation: An error occurred while writing an error message.

Module: Client

115 (X'00000073')

Explanation: The profile could not be opened.

Module: Client

116 (X'00000074')

Explanation: The profile file could not be read.

Module: Client

117 (X'00000075')

Explanation: The profile end-of-file was unexpectedly reached.

Module: Client

118 (X'00000076')

Explanation: The conversation did not initialize. Some possible causes are:

- You encountered a RACF violation.
- The TP is not active.
- APPC is not running.
- You might have incorrect definitions in CM or VTAM®, or APPC or TP setup.
- You might be trying to use APPC to connect an AIX requester that only supports TCP/IP, an OS/2 requester that only supports TCP/IP, a Windows NT requester that only support TCP/IP, a Solaris requester, or an HP requester to MVS by using the IDBSYMDESTNAME parameter in your database profile. This parameter cannot be used with an AIX requester that only supports TCP/IP, an OS/2 requester that only supports TCP/IP, a Windows NT requester that only supports TCP/IP, a Solaris requester, or an HP requester.
- You might be trying to use TCP/IP to connect an OS/2 requester that only supports APPC to MVS by using the IDBSERVERHOST parameter in your database profile. This parameter cannot be used with an OS/2 requester that only supports APPC.
- TCP/IP may not be running or the network may be down.

Module: Client

119 (X'00000077')

Explanation: IDBSYMDESTNAME in database profile specifies an unrecognized value.

Module: Client

120 (X'00000078')

Explanation: Process ID of caller failed to match process ID of original caller for the transaction sequence.

Module: Client

121 (X'00000079')

Explanation: Effective user ID of calling process failed to match effective user ID of original calling process for the transaction sequence.

Module: Client

122 (X'0000007A')

Explanation: The file /etc/utmp was not found on the client interface host.

Module: Client

123 (X'0000007B')

Explanation: The BOOT_TIME record was not found in /etc/utmp on the client interface host.

Module: Client

124 (X'0000007C')

Explanation: Insufficient space is available in the descriptor table of the calling process.

Module: Client

125 (X'0000007D')

Explanation: Insufficient space is available in a file system of the calling process.

Module: Client

126 (X'0000007E')

Explanation: The user's quota of disk blocks or i-nodes on a file system is exhausted.

Module: Client

127 (X'0000007F')

Explanation: The root or current directory of the calling process is located in a virtual file system that is not mounted.

Module: Client

128 (X'00000080')

Explanation: A system-wide or per-user limit on the number of processes prevented creation of a process by the client interface.

Module: Client

129 (X'00000081')

Explanation: Insufficient memory is available for allocation by the client interface.

Module: Client

130 (X'00000082')

Explanation: A system-imposed limit on the number of shared memory identifiers prevented creation of a shared memory segment by the client interface.

Module: Client

131 (X'00000083')

Explanation: Insufficient physical memory was available on the client interface host.

Module: Client

132 (X'00000084')

Explanation: A system-imposed limit on the number of shared memory segments attached to a process prevented attachment of a shared memory segment by the client interface.

Module: Client

133 (X'00000085')

Explanation: Insufficient data space was available in memory for a shared memory segment on the client interface host.

Module: Client

134 (X'00000086')

Explanation: The requester failed to create a process.

Module: Client

135 (X'00000087')

Explanation: Insufficient memory was available for the requester.

Module: Client

137 (X'00000089')

Explanation: The number of conversation managers would have exceeded the requester's limit.

Module: Client

138 (X'0000008A')

Explanation: The requester experienced an error during client logon to the conversation process manager.

Module: Client

139 (X'0000008B')

Explanation: Reserved

140 (X'0000008C')

Explanation: The system file table is full.

Module: Client

141 (X'0000008D')

Explanation: A directory cannot be extended to contain a new file.

Module: Client

142 (X'0000008E')

Explanation: Data conversion between code sets (code pages) failed.

Module: Client

143 (X'0000008F')

Explanation: No match for the specified server service name or alias was found.

If this is a HLAPI/UNIX client application, specify a valid server service name in the `etc/services` file on the requester host. If this is a HLAPI/2 or a HLAPI/NT client application, specify a valid server service name in the `services` file in the `etc` subdirectory. If this is a HLAPI/USS client application, specify a valid server service name in the `/etc/services` file or `hlq.ETC.SERVICES` data set, whichever you use, on the requester host. Service names are case-sensitive.

Module: Client

144 (X'00000090')

Explanation: The specified server host name could not be resolved. The server host name must be a valid IP address in the dotted-decimal format or a valid host name. If you specify a host name, the host name must be resolvable.

Module: Client

145 (X'00000091')

Explanation: The server host is not available. The server must be started before a requester can access it.

Verify that you specified the correct server host name and server service name. If this is a HLAPI/UNIX client application, verify `etc/services` on the requester host you are accessing contains the server service name and that the port number associated with it is the one designated for the Tivoli Information Management for z/OS MRES with TCP/IP on the server host. If this is a HLAPI/2 or a HLAPI/NT client application, verify the `services` file in the `etc` subdirectory contains the server service name and that the port number associated with it is the one designated for the Tivoli Information Management for z/OS MRES with TCP/IP on the server host. If this is a HLAPI/USS client application, verify `/etc/services` or `hlq.ETC.SERVICES`, whichever you use, on the requester host you are accessing, contains the server service name and that the port number associated with it is the one designated for the Tivoli Information Management for z/OS MRES with TCP/IP on the server host.

Start the MRES with TCP/IP.

Module: Client

146 (X'00000092')

Explanation: Reserved

147 (X'00000093')

Explanation: Reserved

148 (X'00000094')

Explanation: Reserved

149 (X'00000095')

Explanation: Reserved

150 (X'00000096')

Explanation: The beginning match number is greater than the number of matches in the search.

Module: LLAPI

151 (X'00000097')

Explanation: The value in PICARHIT indicates to use an existing search, but no search ID is specified in PICASRID.

Module: LLAPI

152 (X'00000098')

Explanation: The product s-word or visible phrase was not found for a create transaction. The data view record did not contain the product s-word.

Module: LLAPI

153 (X'00000099')

Explanation: The application ID is not authorized to update the record. The record has an owning or transfer-to class and the application ID must be running in one of these classes or in the master privilege class.

Module: LLAPI

154 (X'0000009A')

Explanation: Record access panel error on a create transaction. Either the record access panel cannot be found in the PIDT generated by the data view, it cannot be loaded, or it is not a control panel.

Module: LLAPI

155 (X'0000009B')

Explanation: The SDLDS is full and must be offloaded.

Module: LLAPI

156 (X'0000009C')

Explanation: The privilege class used with the change record approval transaction is not on the pending approval list for the change record. One of two cases exists:

- The current privilege class has already approved or rejected the change.
- The current privilege class was either never entered as change approver or else it has been removed.

Module: LLAPI

157 (X'0000009D')

Explanation: The data view record ID specified in the PICATABN field could not be found on the data model database.

Module: LLAPI

158 (X'0000009E')

Explanation: Multiple records exist on the database for the data view record ID specified in the PICATABN field.

Module: LLAPI

159 (X'0000009F')

Explanation: The data view record ID specified in the PICATABN field is busy and cannot be processed.

Module: LLAPI

160 (X'000000A0')

Explanation: The *MVS application user ID* has not been authorized to use the application ID value specified for PICAUSRN. This check has been requested by the specification of the keyword **APISECURITY=ON** in the BLX-SP startup parameters. This reason code occurs with an abend code of 702 at the server.

Module: LLAPI

161 (X'000000A1')

Explanation: When using the HLAPI, if **BYPASS_PANEL_PROCESSING=YES** is specialized at initialization (HL01), data model records must be used for the following HLAPI transactions:

- HL08 -- Create record
- HL09 -- Update record
- HL12 -- Add record relation

When using a LLAPI, if **PICADRIF=Y** is specified at initialization (T001), **PICADMRC=Y** must be specified for the following LLAPI transactions:

- T101 -- Obtain record create resource
- T102 -- Create record
- T103 -- Obtain record update resource
- T105 -- Update record
- T108 -- Obtain add record relation resource
- T109 -- Add record relation

When using a LLAPI, if **PICADRIF=Y** is specified at initialization (T001), **PICADYNM=Y** may not be specified for the following LLAPI transactions:

- T102 -- Create record
- T105 -- Update record

When using a LLAPI, if **PICADMRC=Y** is specified, **PICADYNM=Y** may not be specified for the following LLAPI transactions:

- T100 -- Retrieve record
- T102 -- Create record
- T105 -- Update record

Module: LLAPI

---

162 (X'000000A2')

Explanation: Reserved.

Module: LLAPI

---

163 (X'000000A3')

Explanation: The length of parameter data being passed to a TSP or TSX is greater than 255. User exit BLGYITSP encountered the error. The parameter length is specified in PICAPARL, which is set by an application invoking LLAPI transaction T111 or HLAPI transaction HL14 (parameter data is specified in an input PDB named USER_PARAMETER_DATA). To correct this error, modify the application to specify 255 or fewer characters to be passed to the TSP or TSX being invoked.

Module:

LLAPI

---

164 (X'000000A4')

Explanation: Error invoking an application-specified TSP or TSX on the T111 or HL14 transaction. Common errors are that the TSP or TSX was not found, or that a TSX syntax error occurred and no REXX syntax routine was specified to process it. User exit BLGYITSP encountered the error and has set TSCAFRET and TSCAFRES to further detail the error. Trace the output of TSP BLGAPI00 (if using panel processing) or BLGAPIDI (if using bypass panel processing) to determine the TSCAFRET and TSCAFRES values returned by user exit BLGYITSP.

Module:

LLAPI

---

165 (X'000000A5')

Explanation: A TSP or TSX invoked by the HL14 transaction encountered a syntax error. This reason code is reserved to indicate that a syntax error occurred. The TSP or TSX must capture the syntax error in a syntax routine and set this reason code by calling user exit BLGYAPSR or, in the case of a TSX, by invoking control line SETAPIDATA.

Note: You can use this reason code in your user-written TSPs or TSXs.

Module:

LLAPI

---

166 (X'000000A6')

Explanation: A TSP or TSX invoked by the HL14 transaction encountered a general error. This reason code is reserved for TSPs and TSXs to indicate that a general error occurred. The TSP or TSX must capture the syntax error in a syntax routine and set this reason code by calling user exit BLGYAPSR or, in the case of a TSX, by invoking control line SETAPIDATA. To determine the nature of this problem, look for additional messages which may indicate the specific error encountered. Trace the TSP or TSX being run by the HL14 to gather additional information about the error.

Note: You can use this reason code in your user-written TSPs or TSXs.

Module:

LLAPI

---

167 (X'000000A7')

Explanation: Record access attempted and failed because the record is currently being updated by the Tivoli Service Desk application.

Module: LLAPI

168 (X'000000A8')

Explanation: Record update attempted and failed because the record is currently owned by the Tivoli Service Desk application.

Module: LLAPI

169 (X'000000A9')

Explanation: Record update attempted and failed because the record contains one or more dates that could not be migrated to the new date format.

Module: LLAPI

169 (X'000000A9') – 199 (X'000000C7')

Explanation: Reserved.

Module:

200 (X'000000C8')

Explanation: The MRES API session has been pre-started with parameters which do not match those received on an HL01 transaction from a remote client. The HL01 transaction request is rejected. One of the following inconsistencies has occurred:

- The SESSION_MEMBER names are not the same.
- The DATABASE_ID numbers are not the same.
- The BYPASS_PANEL_PROCESSING values are not the same.

Module: MRES

201 (X'000000C9')

Explanation: Both the preceding HL01 transaction and this transaction do not contain an APPLICATION_ID PDB. The access authority cannot be checked. Therefore, this transaction is rejected.

Module: MRES

202 (X'000000CA')

Explanation: The pre-started API session of an MRES has received two HL01 client transactions without an intervening HL02 transaction. This is not permitted. A client must terminate one API session before initializing a second API session.

Module: MRES

203 (X'000000CB')

Explanation: The first transaction received from a client by an MRES with pre-started API sessions is not the HL01 initialization transaction. This is not permitted. The first transaction received from a client must be the HL01 initialization transaction.

Module: MRES

1000 - 9999

Explanation: Reserved for user TSP (terminal simulator panel) use.

Reason Codes for Return Code=16

All API resources required by the transaction are verified before passing the transaction on to the API subtask for processing. Failure of any of the resource verification steps results in an API ABEND. The following steps are performed:

1. The resource is verified as an API-allocated resource.
2. The allocated size of the resource is verified.
3. The integrity of the resource limitations is verified.

When a severe error occurs, the APIs stop with a return code of 16 and a reason code in the form xxsssuuu where sss is an abend code, uuu is a reason code, and xx are do-not-care positions. The ABEND codes returned by the API are listed below. The reason codes follow based on each ABEND code. These codes are also listed in the *Tivoli Information Management for z/OS Messages and Codes* book along with problem determination information.

ABEND Codes

The following abend codes are returned in the reason code.

Abend Code 700 (X'2BC')

The APIs end with the abend code 700 (X'2BC') when a severe error occurs. “Reason Codes for ABEND Code 700” contains the associated reason codes.

Reason Codes for ABEND Code 700

The following reason codes apply when the abend code is 700 (X'2BC').

004 (X'004')

Explanation: Internal control structures required for transaction processing could not be located.

Module: HLAPI, LLAPI

008 (X'008')

Explanation: Allocation of the LLAPI activity log failed or reallocation of the activity log failed after the spool interval timer expired.

Module: LLAPI

012 (X'00C')

Explanation: Deallocation of the HLAPI activity log failed after the spool interval timer expired, or deallocation of the activity log failed during API termination.

Module: HLAPI

016 (X'010')

Explanation: Allocation of the table data set failed during LLAPI initialization.

Module: LLAPI

020 (X'014')

Explanation: Storage allocation for the table data set buffer failed during API initialization.

Module: LLAPI

024 (X'018')

Explanation: An incorrect parameter list (PICA) was passed to request LLAPI transaction processing.

Module: LLAPI

028 (X'01C')

Explanation: An incorrect internal parameter list was used during LLAPI transaction processing.

Module: LLAPI

032 (X'020')

Explanation: A LLAPI TSP user exit was called when the LLAPI was not active.

Module: LLAPI

036 (X'024')

Explanation: A corrupted PIAT structure was detected during transaction validation.

Module: LLAPI

040 (X'028')

Explanation: A corrupted PIDT structure was detected during transaction validation.

Module: LLAPI

044 (X'02C')

Explanation: A corrupted PIPT structure was detected during transaction validation.

Module: LLAPI

048 (X'030')

Explanation: A corrupted PIRT structure was detected during transaction validation.

Module: LLAPI

052 (X'034')

Explanation: A corrupted response buffer was detected during transaction validation.

Module: LLAPI

056 (X'038')

Explanation: The response data pointer (PIDTDATP) in a PIDT entry contains an address that is outside the response buffer address range.

Module: LLAPI

060 (X'03C')

Explanation: The response data length (PIDTCURL) in a PIDT entry when added to the response data address (PIDTDATP) results in an address that is outside the response buffer address range.

Module: LLAPI

064 (X'040')

Explanation: A corrupted PALT structure was detected.

Module: LLAPI

080 (X'050')

Explanation: An incorrect HICA structure was passed to BLGYHLPI.

Module: HLAPI

092 (X'05C')

Explanation: Initialization transaction from the HLAPI/REXX interface failed, and an attempt to stop the HLAPI from REXX also failed. The HLAPI/REXX interface is unable to clean up the Tivoli Information Management for z/OS environment.

Module: HLAPI/REXX

096 (X'060')

Explanation: HLAPI/REXX interface encountered an error and has attempted to write information about the error back to the REXX program. The attempt to return this information failed due to a variable access error.

Module: HLAPI/REXX

100 (X'064')

Explanation: The value of storage referenced by BLG_ENVP was corrupted. Possible reasons: the value of BLG_ENVP is not a valid pointer, the storage itself contains data that is not valid, or the HLAPI is corrupted.

Module: HLAPI/REXX

Abend Code 702 (X'2BE')

A connection request from the API has been terminated. The MVS userid of the API task attempting to connect to the BLX-SP has not been authorized to use the application ID value specified in the PICAUSRN field of the LLAPI PICA control block.

Abend Code 706 (X'2C2')

A HLAPI client ends with the abend code 706 (X'2C2') when a severe error occurs while initializing a conversation with either a Remote Environment Server (RES) or Multiclient Remote Environment Server (MRES). “Reason Codes for ABEND Code 706” contains the associated reason codes.

Reason Codes for ABEND Code 706

The following reason codes apply when the abend code is 706 (X'2C2').

001 (X'001')

Explanation: Initialization of the BLX environment failed during Remote Environment Server initialization.

Module: Server

002 (X'002')

Explanation: A failure occurred while attempting to load the HLAPI interface module during Remote Environment Server initialization.

Module: Server

003 (X'003')

Explanation: A failure occurred while establishing the conversation between the remote system and the Remote Environment Server.

Module: Server

004 (X'004')

Explanation: A failure occurred while initializing the Remote Environment Server.

Module: Server

008 (X'008')

Explanation: A failure occurred while initializing the Remote Environment Server.

Module: Server

Abend Code 707 (X'2C3')

The APIs end with the abend code 707 (X'2C3') when a severe error occurs while initializing a Multiclient Remote Environment Server (MRES). The reason code provided in a preceding message or in register 15 of the dump identifies the specific type of error encountered. “Reason Codes for ABEND Code 707” contains the associated reason codes.

Reason Codes for ABEND Code 707

The following reason codes apply when the abend code is 707 (X'2C3').

004 (X'004')

Explanation: The BLX environment initialization failed.

Module: Server

008 (X'008')

Explanation: The error recovery initialization failed.

Module: Server

012 (X'00C')

Explanation: Insufficient storage was available for initialization.

Module: Server

016 (X'010')

Explanation: A failure occurred while initializing the Multiclient Remote Environment Server control block.

Module: Server

020 (X'014')

Explanation: The MRES communication manager initialization failed.

Module: Server

024 (X'018')

Explanation: The start of the MRES communication manager failed.

Module: Server

028 (X'01C')

Explanation: The MRES operator interface initialization failed.

Module: Server

032 (X'020')

Explanation: The APPC Register_For_Allocates service failed.

Module: Server

036 (X'024')

Explanation: An error was detected while attempting to access the parameters data set allocated with the BLMYPRM DD statement.

Module: Server

040 (X'028')

Explanation: An error was detected while attempting to open the parameters data.

Module: Server

044 (X'02C')

Explanation: An error was detected while attempting to read the parameters data set.

Module: Server

Abend Code 708 (X'2C4').

A HLAPI client ends with the abend code 708 (X'2C4') when the Multiclient Remote Environment Server (MRES) receives a command that is not STOP. This abend code has no reason codes associated with it.

Origin:

Server

Reason Codes for Return Code=20

The following reason codes apply when the return code is 20 (X'014').

1 (X'00000001')

Explanation: An unexpected system error occurred. If this is a HLAPI/CICS client application, see the HLAPI/CICS log file (transient data queue BLML) for more information. If this is a HLAPI/2 or HLAPI/NT client application, see the system `idbprobe.log` file for more information. If this is a HLAPI/UNIX client application, see the system `idbprobe.log` file for more information. If this is a HLAPI/USS client application, see the system `blmprobe.log` file for more information.

Module: Client

2 (X'00000002')

Explanation: The conversation unexpectedly ended.

Module: Client

3 (X'00000003')

Explanation: The conversation within the requester has stopped processing.

Module: Client

4 (X'00000004')

Explanation: The requester has stopped processing.

Module: Client

5 (X'00000005')

Explanation: The server has encountered an unexpected error. If this is a HLAPI/CICS client application, see the HLAPI/CICS log file (transient data queue BLML) for more information. If this is a HLAPI/2 or HLAPI/NT client application, see the system `idbprobe.log` file for more information. If this is a HLAPI/UNIX client application, see the system `idbprobe.log` file for more information. If this is a HLAPI/USS client application, see the system `blmprobe.log` file for more information.

Module: Server

6 (X'00000006')

Explanation: The requester has been restarted. Previous conversations were dropped.

Module: Client

7 (X'00000007')

Explanation: The requester communicated a general error to the client interface. See the system idbprobe.log file for more information. If this is a HLAPI/USS client application, see the system `blmprobe.log` file for more information.

Module: Client

HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, and REXX HLAPI/USS Return Codes

The following return codes are returned by HLAPI/REXX and by the REXX client interfaces REXX HLAPI/2, REXX HLAPI/AIX, and REXX HLAPI/USS.

- HLAPI/REXX and REXX HLAPI/USS return them in the RC variable.
- REXX HLAPI/2 and REXX HLAPI/AIX return them in the RESULT variable.

Errors that occur before HLAPI is called by HLAPI/REXX, before HLAPI/2 is called by REXX HLAPI/2, before HLAPI/AIX is called by REXX HLAPI/AIX, and before HLAPI/USS is called by REXX HLAPI/USS, cause processing to stop. If a nonzero return code is returned by the HLAPI, HLAPI/2, HLAPI/AIX, or HLAPI/USS, processing continues and the associated REXX HLAPI (HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, REXX HLAPI/USS) attempts to return all output information. If an error occurs while returning that information, processing stops and the return code reflects that error. However, BLG_RC and BLG_REAS are set to indicate that there was a HLAPI, HLAPI/2 HLAPI/AIX, or HLAPI/USS error unless the associated REXX HLAPI found an error when setting one of those variables.

Each return code is accompanied by an explanation and an origin. The origin indicates the particular REXX HLAPI that produces the return code. The following terms are used to identify the origin of a reason code:

HLAPI/REXX
Code returned by the HLAPI/REXX

REXX HLAPI/2
Code returned by the REXX HLAPI/2

REXX HLAPI/AIX
Code returned by the REXX HLAPI/AIX

REXX HLAPI/USS
Code returned by the REXX HLAPI/USS

0

Explanation: Indicates that the transaction completed successfully.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

4

Explanation: BLGYRXM was started incorrectly. When REXX starts an external routine, it passes a pointer to the environment block that represents the REXX language processing environment. This pointer was 0. BLGYRXM must be started from a REXX exec. Refer to *TSO Extensions Version 2 REXX Reference* for more information about the language processing environment.

Module: HLAPI/REXX

8

Explanation: Parameter list not valid (*transaction-name* must be specified).

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX or REXX HLAPI/USS

12

Explanation: Passed array stem is greater than 32 characters. The *control*, *input*, or *output* parameter is greater than 32 characters.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

16

Explanation: Transaction name specified is not valid. “HLAPI/REXX Interface Calls” on page 241 contains information on specifying the transaction name.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

20

Explanation: REXX HLAPI interface environment pointer not valid. The REXX HLAPI interface uses REXX variable BLG_ENVP to store a pointer to a control storage area that the REXX HLAPI interface uses. Possible errors: you are attempting a transaction which is not an initialization transaction and the variable doesn’t exist or the variable was null.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

24

Explanation: Data is too long. BLG_VARNAME is set to the name of the control variable that contains the incorrect data. For a list of maximum input lengths, see “Maximum Input Lengths” on page 248.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

28

Explanation: Variable does not exist or has null value. BLG_VARNAME contains the name of the variable. Possible causes: missing or null *Input*.0, *control*.0, *text-name*.0, control variables, or input variables.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

32

Explanation: Alias name specified not valid. An input variable name was greater than 32 characters. BLG_VARNAME contains the name of the variable that was used to specify the incorrect name.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

36

Explanation: Variable contains a numeric string that is not valid. The value was not all numbers, or was less than 0 or greater than 2 147 483 647. BLG_VARNAME contains the name of the variable whose value caused the error. Possible causes:

- *control*.0 is not between 0 and 30
- *Input*.0, FREEFORM.0, or *text-name*.0 is not less than or equal to 32 767
- *text-name.?width* is not less than or equal to 132

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

40

Explanation: Error initializing a REXX variable. Possible causes: environment problem, output stem specified not valid (REXX variable name not valid), or internal processing error.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

44

Explanation: *Input*.0 contains a numeric value greater than 0 and the REXX HLAPI interface can find no inputs.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

100

Explanation: Error allocating storage for either the REXX variable names and values, or for the PDBs for the HLAPI.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

104

Explanation:

- For HLAPI/REXX and REXX HLAPI/USS, this is the error returned by the REXX variable access service IRXEXCOM. BLG_IRXEXCOM_RC is set to the return code (-1, -2, or 28). Refer to the *TSO Extensions Version 2 REXX Reference* for information about IRXEXCOM.
- For REXX HLAPI/2, this is the error returned by the OS/2 REXX variable pool access service. BLG_REXXVAR_POOL_RC is set to the return code.
- For REXX HLAPI/AIX, this is the error returned by the AIX REXX/6000 variable pool access service. BLG_REXXVAR_POOL_RC is set to the return code.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

108

Explanation: Error reading input variable. BLG_VARNAME contains the name of the variable. BLG_SHVRET contains a 1-character hexadecimal return code SHVRET that indicates the error in accessing the variable.

- For HLAPI/REXX and REXX HLAPI/USS, BLG_IRXEXCOM_RC contains the return code from the REXX variable access service IRXEXCOM. Refer to *TSO Extensions Version 2 REXX Reference* for information about IRXEXCOM.
- For REXX HLAPI/2, BLG_REXXVAR_POOL_RC contains the return code from the OS/2 REXX variable pool access service.
- For REXX HLAPI/AIX, BLG_REXXVAR_POOL_RC contains the return code from the AIX REXX/6000 variable pool access service.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

112

Explanation: Error setting output variable. BLG_VARNAME contains the name of the variable. BLG_SHVRET contains a 1-character hexadecimal return code SHVRET that indicates the error in accessing the variable.

- For HLAPI/REXX and REXX HLAPI/USS, BLG_IRXEXCOM_RC contains the return code from the REXX variable access service IRXEXCOM. Refer to *TSO Extensions Version 2 REXX Reference* for information about IRXEXCOM.
- For REXX HLAPI/2, BLG_REXXVAR_POOL_RC contains the return code from the OS/2 REXX variable pool access service.
- For REXX HLAPI/AIX, BLG_REXXVAR_POOL_RC contains the return code from the AIX REXX/6000 variable pool access service.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

116

Explanation: Error dropping output variable. BLG_VARNAME contains the name of the variable. BLG_SHVRET contains a 1-character hexadecimal return code SHVRET that indicates the error in accessing the variable.

- For HLAPI/REXX and REXX HLAPI/USS, BLG_IRXEXCOM_RC contains the return code from the REXX variable access service IRXEXCOM. Refer to *TSO Extensions Version 2 REXX Reference* for information about IRXEXCOM.
- For REXX HLAPI/2, BLG_REXXVAR_POOL_RC contains the return code from the OS/2 REXX variable pool access service.
- For REXX HLAPI/AIX, BLG_REXXVAR_POOL_RC contains the return code from the AIX REXX/6000 variable pool access service.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

200

Explanation: Nonzero return code from the HLAPI. BLG_RC and BLG_REAS contain the return and reason code. Use these codes to interpret the HLAPI error. Input errors, output errors, or both might have been set. BLG_MSGS and BLG_ERRCODE might also contain information about the error.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or REXX HLAPI/USS

204

Explanation:

- For HLAPI/REXX, error occurred loading or linking to the HLAPI server (BLGYHLPI).
- For REXX HLAPI/2, error occurred calling HLAPI/2 (IDBTransactionSubmit), in which case the transaction could not be completed. BLG_HLAPI2_RC contains the return code from HLAPI/2.
- For REXX HLAPI/AIX, error occurred calling HLAPI/AIX (IDBTransactionSubmit), in which case the transaction could not be completed. BLG_HLAPIAIX_RC contains the return code from HLAPI/AIX.
- For REXX HLAPI/USS, error occurred calling HLAPI/USS (IDBTransactionSubmit), in which case the transaction could not be completed. BLG_HLAPI/USS_RC contains the return code from HLAPI/USS.

Module: HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, or HLAPI/USS

C

Terminal Simulator Panels

The LLAPI provides the following TSPs for the API subtask. The HLAPI uses the LLAPI to perform its tasks, so indirectly it also uses these TSPs.

If you are using the panels that Tivoli Information Management for z/OS supplies for entering problem, change, or configuration records, do not modify these TSPs. If you have changed the panels supplied by Tivoli Information Management for z/OS or you are processing other types of records, then some modifications may be necessary for these TSPs to run without error. To make the modifications correctly, you must understand the flow of these TSPs. You may need to make modifications if the application selection is different, if the entry selection to enter records is different, or if the file selection is different. There might be other modifications necessary depending on the panel flows.

BLGAPI00–LLAPI Router TSP for Panel Processing

This panel is the controlling panel for the LLAPI. It performs the following steps:

1. Sets up the LLAPI environment and waits for a transaction
2. Tests transaction code and links to transaction TSP
3. On return, branches to API control user exit
4. Quits or frees the LLAPI environment on error or T002 transaction.

TERMINAL SIMULATOR PANEL COMMON CONTROL FLOW DATA

LINE NUM	FUNCTION NAME	LABEL NAME	LITERAL/TEST DATA	PANEL NAME	FUNCTION EXIT	FIELD NAME	WORD OCCUR	APPLY NOT	GET VAR	CASE-SENS
1	LABEL		*** INITIALIZE AND SELECT							
2	LABEL		*** INFO/MANAGEMENT							
3	ADDDATA		;INIT,3,2						NO	
4	PROCESS	INITERR								
5	LABEL	APIWAIT	*** WAIT FOR NEXT TRANSACTION							
6	USEREXIT		*** CALL API CONTROL USER EXIT		BLGYAPCP		NEXT	NO	NO	
7	LABEL		TEST FOR TERMINATION							
8	TESTFIELD	LINKT002	T002			TSCAUFLD		NO	NO	NO
9	LABEL		TEST FOR CREATE TRANSACTION							
10	TESTFIELD	LINKT102	T102			TSCAUFLD		NO	NO	NO
11	LABEL		TEST FOR UPDATE TRANSACTION							
12	TESTFIELD	LINKT105	T105			TSCAUFLD		NO	NO	NO
13	LABEL		TEST FOR ADD RELATION XACTION							
14	TESTFIELD	LINKT109	T109			TSCAUFLD		NO	NO	NO
15	LABEL		TEST FOR DELETE TRANSACTION							
16	TESTFIELD	LINKT110	T110			TSCAUFLD		NO	NO	NO
17	LABEL		TEST FOR INVOKE USER TSP							
18	TESTFIELD	LINKT111	T111			TSCAUFLD		NO	NO	NO
19	BRANCH	APIWAIT								
20	LABEL	LINKT102	LINK TO CREATE TSP							
21	LINK			BLGAPI02						
22	LABEL		REMOVE FOLLOWING BRANCH TO							
23	LABEL		ENABLE THE AUTOBRIDGE							
24	LABEL		POSTPROCESSOR							
25	BRANCH	DISABLED								
26	USEREXIT		API POSTPROCESSOR NOTIFY		EYMSP010		NEXT	NO	NO	
27	LABEL	DISABLED								

```
28  BRANCH      APIWAIT
29  LABEL       LINKT105  LINK TO UPDATE TSP
30  LINK                                          BLGAPI05
31  BRANCH      APIWAIT
32  LABEL       LINKT109  LINK TO ADD DATA TSP
33  LINK                                          BLGAPI09
34  BRANCH      APIWAIT
35  LABEL       LINKT110  LINK TO DELETE TSP
36  LINK                                          BLGAPI10
37  BRANCH      APIWAIT
38  LABEL       LINKT111  INVOKE OR LINK TO USER TSP/TSX
39  USEREXIT                                                  BLGYITSP            NEXT     NO    NO
40  TESTFIELD   APIWAIT   0                                             TSCAFRET            NO    NO   NO
41  TESTFIELD   APIWAIT   4                                             TSCAFRET            YES   NO   NO
42  LABEL               REPLACE THIS WITH LINK STATEMENT
43  LABEL               TO BRANCH TO HARDCODED TSP/TSX
44  BRANCH      APIWAIT
45  LABEL       INITERR   PRINT THE MESSAGES AND SCREEN
46  PRINT
47  LABEL       LINKT002
48  ADDDATA             ;QUIT                                                                   NO
49  PROCESS     ENDIT
50  LABEL       ENDIT     LEAVE THE TSP
51  RETURN
```

BLGAPI02–LLAPI Create Record TSP for Panel Processing

This TSP includes the following functions for special processing. All of these functions are shipped disabled.

- Use of a dynamic PIDT (PIDTUSEF=D)
- History data processing (PICAHIST=Y)
- Text audit data processing (PICATXAU=Y).

You can enable processing for these functions by removing the DISABLED branch on line 16.

If you enable the special processing functions, the TSP checks whether the application program has database administration authority. If the application does not have the required authority, the request fails, and a reason code is put in PICAREAS.

You can change the authority required for special processing by changing the s-word indexes on line 32 (MOVEVAR). Refer to the section “S–Word Authorization Codes” in the *Tivoli Information Management for z/OS Panel Modification Facility Guide* for a list of s-word indexes.

This TSP performs the following steps:

1. Runs the IRC `;INIT,3,2` to go to the Tivoli Information Management for z/OS PRIMARY OPTIONS MENU (selection 5 is for Entry). Ensure that this IRC flows to a panel with a selection that runs program exit BLG01050.
2. Calls BLGYAPGP to:
   a. Take the specified record type s-word from the create PIDT and look for a match in control panel BLG1AACP. If the TSP does not find a match, the TSP ends with an error. If a match is found, the TSP saves the panel named in that control line. (The panel is the summary panel for that record type with a 9 selection to file.) The summary panel is saved in the TSCA variable area.

      If you are creating a record of your own type, you must build a PIDT with the correct record type s-word and update panel BLG1AACP using the record type s-word for your record and its corresponding create summary panel.

b. Check whether special processing (history data, text audit data, or dynamic PIDT) is requested.

3. If special processing is not requested:

   a. Issues 5 for Entry (flows to create summary panel read from BLG1AACP). Ensure that this IRC runs program exit BLG01050. If this program exit runs under the LLAPI, your regular create panels are bypassed because BLG01050 flows to the create summary panel obtained from BLG1AACP, not the normal target of the selection.

   b. Adds data (user exit BLGYAPBR).

   c. Issues 9 to file the record. File time program exits, TSPs, and TSXs are started. After the file processing has completed, the BLGAPI02 TSP looks for the message BLG03058I on the TSP saved message chain, indicating the record was filed successfully. If the message is found, the TSP returns a successful return code to the application, even if mail notification processing is invoked for the record.

4. If special processing is requested and the disabling branch has not been removed, this TSP puts a reason code in TSCAFRES and exits.

5. If special processing is requested and the disabling branch has been removed, performs the following steps:

   a. Issues 5 for Entry (flows to create summary panel read from BLG1AACP). Ensure that this IRC runs program exit BLG01050. If this program exit runs under the LLAPI, your regular create panels are bypassed because BLG01050 flows to the create summary panel obtained from BLG1AACP, not the normal target of the selection.

   b. Checks for database administration authority. If the application does not have the required authority, puts a reason code in TSCAFRES and exits.

   c. Adds data (user exit BLGYAPBR).

   d. Issues 9 to file the record. File time program exits, TSPs, and TSXs are started. After the file processing has completed, the BLGAPI02 TSP looks for the message BLG03058I on the TSP saved message chain, indicating the record was filed successfully. If the message is found, the TSP returns a successful return code to the application, even if mail notification processing is invoked for the record.

      If selection 9 does not file the record from your summary panel, modify the TSP to make the correct selection.

This is TSP BLGAPI02.

```
  TERMINAL SIMULATOR PANEL COMMON CONTROL FLOW DATA

 LINE  FUNCTION     LABEL        LITERAL/TEST                PANEL    FUNCTION   FIELD     WORD    APPLY  GET  CASE-
 NUM     NAME       NAME            DATA                     NAME       EXIT     NAME      OCCUR    NOT   VAR  SENS
 ---  ----------  --------  --------------------------------  --------  --------  --------  ------  ---   ---  ---

   1  LABEL       BLGAPI02  *** API CREATE RECORD
   2  ADDDATA               ;INIT,3,2                                                              NO
   3  PROCESS     EXERROR
   4  LABEL                 *** CHECK AUTH AND GET CREATE
   5  LABEL                 *** SUMMARY PANEL NAME
   6  USEREXIT              CALL BLGYAPGP                       BLGYAPGP             NEXT     NO    NO
   7  LABEL                 *** BRANCH IF NO ERRORS AND NO
   8  LABEL                 *** SPECIAL PROCESSING REQUESTED
   9  TESTFIELD   CREATE    0                                              TSCAFRET            NO    NO   NO
  10  LABEL                 *** BRANCH IF ERRORS WERE FOUND
  11  TESTFIELD   ERROR     0                                              TSCAFRES            NO    NO   NO
  12  LABEL                 *** REMOVE FOLLOWING BRANCH TO
```

```
13  LABEL               *** ENABLE SPECIAL PROCESSING
14  LABEL               *** FOR HISTORY DATA, FREEFORM
15  LABEL               *** TEXT, AND DYNAMIC PIDT
16  BRANCH      DISABLED
17  LABEL               *** SAVE PANEL NAME, SET REASON
18  LABEL               *** CODE TO ZERO, RESTORE PANEL
19  SETFIELD                                                    TSCAUFLD                YES
20  MOVEVAR             00
21  USEREXIT            CALL BLGYAPSR - RESET REASON            BLGYAPSR        NEXT    NO    NO
22  MOVEVAR                                                     TSCAUFLD
23  LABEL               *** INITIATE CREATE BEFORE
24  LABEL               *** CHECKING AUTHORIZATION FOR
25  LABEL               *** SPECIAL PROCESSING
26  ADDDATA             5                                                               NO
27  PROCESS     BADCR
28  LABEL               *** TEST TSCACPNL TO VAR. DATA
29  TESTFIELD   BADCR                                           TSCACPNL        YES     YES   NO
30  LABEL               *** TO BEGIN AUTHORIZATION CHECK
31  LABEL               *** CALL USER EXIT BLGJAUTH
32  MOVEVAR             0AA1
33  USEREXIT            CALL BLGJAUTH - CHECK AUTH              BLGJAUTH        NEXT    NO    NO
34  TESTFIELD   CNTLBLK 16                                      TSCAFRET        NO      NO    NO
35  TESTFIELD   NOTAUTH 0                                       TSCAFRET        YES     NO    NO
36  BRANCH      COLLECT
37  LABEL       CREATE  *** INITIATE CREATE WHEN SPECIAL
38  LABEL               *** PROCESSING IS NOT REQUESTED
39  ADDDATA             5                                                               NO
40  PROCESS     BADCR
41  LABEL               *** TEST TSCACPNL TO VAR. DATA
42  TESTFIELD   BADCR                                           TSCACPNL        YES     YES   NO
43  LABEL       COLLECT *** COLLECT API DATA INTERNALLY
44  USEREXIT            CALL BLGYAPBR - COLLECT DATA            BLGYAPBR        NEXT    NO    NO
45  TESTFIELD   ERROR   4                                       TSCAFRET        NO      NO    NO
46  LABEL               *** FILE THE NEW RECORD
47  ADDDATA             9                                                               NO
48  PROCESS     EXERROR
49  TESTFLOW    FMSGMISS                                                        YES     NO
50  LABEL       ERROR
51  LABEL       EXIT
52  RETURN
53  LABEL       BADCR   ** SET COMMUNICATION AREA REASON
54  MOVEVAR             28
55  USEREXIT            CALL BLGYAPSR - SET REASON              BLGYAPSR        NEXT    NO    NO
56  BRANCH      EXIT
57  LABEL       FMSGMISS ** FILE MSG NOT ON CURRENT CHAIN
58  TESTFIELD   EXERROR 4                                       TSCAFRET        YES     NO    NO
59  TESTFIELD   EXERROR 4                                       TSCAFRES        YES     NO    NO
60  BRANCH      EXIT
61  LABEL       EXERROR ** SET COMMUNICATION AREA REASON
62  MOVEVAR             03
63  USEREXIT            CALL BLGYAPSR - SET PIVT REASON         BLGYAPSR        NEXT    NO    NO
64  BRANCH      EXIT
65  LABEL       DISABLED ** SET COMMUNICATION AREA REASON
66  MOVEVAR             74
67  USEREXIT            CALL BLGYAPSR - SET PIVT REASON         BLGYAPSR        NEXT    NO    NO
68  BRANCH      EXIT
69  LABEL       CNTLBLK ** SET COMMUNICATION AREA REASON
70  MOVEVAR             82
71  USEREXIT            CALL BLGYAPSR - SET PIVT REASON         BLGYAPSR        NEXT    NO    NO
72  BRANCH      EXIT
73  LABEL       NOTAUTH ** SET COMMUNICATION AREA REASON
74  MOVEVAR             92
75  USEREXIT            CALL BLGYAPSR - SET PIVT REASON         BLGYAPSR        NEXT    NO    NO
76  BRANCH      EXIT
```

BLGAPI05–LLAPI Update Record TSP for Panel Processing

This TSP includes the following functions for special processing. All of these functions are shipped disabled.

- Use of a dynamic PIDT (PIDTUSEF=D)
- History data processing (PICAHIST=Y)
- Text audit data processing (PICATXAU=Y).

You can enable processing for these functions by removing the DISABLED branch on line 17.

If you enable the special processing functions, the TSP checks whether the application program has database administration authority. If the application does not have the required authority, the request fails, and a reason code is put in TSCAFRES.

You can change the authority required for special processing by changing the s-word index on line 22 (MOVEVAR). Refer to the section "S–Word Authorization Codes" in the *Tivoli Information Management for z/OS Panel Modification Facility Guide* for a list of s-word indexes.

This TSP performs the following:

1. Starts `;INIT,3,2` to go to the Tivoli Information Management for z/OS PRIMARY OPTIONS MENU
2. Calls BLGYAPGP to:
   a. Access panel BLG1AAUP and find the control line containing the record type s-word from the update PIDT. If the TSP does not find a match, the TSP ends with an error. If the TSP finds a match, it saves the associated update summary panel in the TSCA variable area.

      If you are updating a record of your own type, you must build a PIDT with the correct record type s-word and update panel BLG1AAUP using the record type s-word for your record and its corresponding update summary panel.

   b. Check whether special processing (history data, text audit data, or dynamic PIDT) is requested.
3. If special processing is requested:
   a. Checks whether the disabling branch has been removed. If it has not been removed, puts a reason code in TSCAFRES and exits.
   b. Checks for database administration authority. If the application does not have the required authority, puts a reason code in TSCAFRES and exits.
4. Issues `;UPDATE,1,5,2,update-rnid,,` to update the record.
5. Calls BLGYAPBU to insert the record number for `update-rnid`.
6. Flows to the normal update summary panel or to the panel specified in BLG1AAUP if the authorization code field in the BLG1AAUP control line is set to 0001.
7. Verifies that the current panel (update summary panel) is the same as the update summary panel read from BLG1AAUP. If the TSP does not find a match between the current panel name and the panel name read from BLG1AAUP, the TSP ends with an error. Possible causes of this error are specifying the wrong summary panel or updating a record having a type other than that specified in the PIDT.
8. Verifies that the correct record is updated.
9. Adds data (user exit BLGYAPBR).
10. Issues 9 to file the record. File time program exits, TSPs, and TSXs are started. After the file processing has completed, the BLGAPI02 TSP looks for the message BLG03058I on the TSP saved message chain, indicating the record was filed successfully. If the message is found, the TSP returns a successful return code to the application, even if mail notification processing is invoked for the record.

    If selection 9 does not file the record from your summary panel, modify the TSP to make the correct selection.

This is TSP BLGAPI05.

```
TERMINAL SIMULATOR PANEL COMMON CONTROL FLOW DATA

LINE  FUNCTION     LABEL        LITERAL/TEST                    PANEL    FUNCTION   FIELD     WORD    APPLY  GET  CASE-
NUM     NAME       NAME            DATA                         NAME      EXIT     NAME     OCCUR    NOT   VAR  SENS
---  ----------  --------  -------------------------------  --------  --------  --------  ------  ---  ---  ---

  1  LABEL       BLGAPI05  *** API UPDATE RECORD
  2  ADDDATA               ;INIT,3,2                                                                    NO
  3  PROCESS     EXERROR
  4  LABEL                 *** CHECK AUTH AND GET UPDATE
  5  LABEL                 *** SUMMARY PANEL NAME
  6  USEREXIT              CALL BLGYAPGP                      BLGYAPGP             NEXT     NO    NO
  7  LABEL                 *** BRANCH IF NO ERRORS AND NO
  8  LABEL                 *** SPECIAL PROCESSING REQUESTED
  9  TESTFIELD   SAVEPNLN  0                                                TSCAFRET          NO    NO   NO
 10  LABEL                 *** BRANCH IF ERRORS WERE FOUND
 11  TESTFIELD   CANCEL    0                                                TSCAFRES          NO    NO   NO
 12  SETFIELD                                                               TSCAUFLD                YES
 13  LABEL                 *** REMOVE FOLLOWING BRANCH TO
 14  LABEL                 *** ENABLE SPECIAL PROCESSING
 15  LABEL                 *** FOR HISTORY DATA, FREEFORM
 16  LABEL                 *** TEXT, AND DYNAMIC PIDT
 17  BRANCH      DISABLED
 18  MOVEVAR               00
 19  USEREXIT              CALL BLGYAPSR - RESET REASON       BLGYAPSR             NEXT     NO    NO
 20  LABEL                 *** TO BEGIN AUTHORIZATION CHECK
 21  LABEL                 *** CALL USER EXIT BLGJAUTH
 22  MOVEVAR               0AA1
 23  USEREXIT              CALL BLGJAUTH - CHECK AUTH         BLGJAUTH             NEXT     NO    NO
 24  TESTFIELD   CNTLBLK   16                                               TSCAFRET          NO    NO   NO
 25  TESTFIELD   NOTAUTH   0                                                TSCAFRET          YES   NO   NO
 26  LABEL       UPDATER   *** COLLECT INITIAL IRC
 27  MOVEVAR               ;UPDATE,1,5,2,
 28  LABEL                 *** SUFFIX IRC WITH RECORD ID
 29  USEREXIT              CALL BLGYAPBU                      BLGYAPBU             NEXT     NO    NO
 30  TESTFIELD   ERROR     4                                                TSCAFRET          NO    NO   NO
 31  MOVEVAR                ,,
 32  ADDDATA                                                                                     YES
 33  PROCESS     EXERROR
 34  LABEL                 *** CHECKED OUT TO ANOTHER?
 35  TESTFLOW    CHECKOUT                                                                    NO    NO
 36  TESTFLOW    CHECKOUT                                                                    NO    NO
 37  LABEL                 *** DO SUMMARY PANELS MATCH
 38  MOVEVAR                                                                TSCAUFLD
 39  TESTFIELD   BADUPD                                                     TSCACPNL          YES   YES  NO
 40  LABEL       CHECKRCD  *** ENSURE THAT THE CORRECT
 41  LABEL                 *** RECORD IS BEING UPDATED
 42  USEREXIT              CALL BLGYAPUP                      BLGYAPUP             NEXT     NO    NO
 43  TESTFIELD   BADUPD    4                                                TSCAFRET          NO    NO   NO
 44  LABEL                 *** CONVERT API DATA TO INTERNAL
 45  USEREXIT              CALL BLGYAPBR - ADD DATA           BLGYAPBR             NEXT     NO    NO
 46  TESTFIELD   CANCEL    4                                                TSCAFRET          NO    NO   NO
 47  LABEL                 *** FILE THE RECORD
 48  ADDDATA               9                                                                           NO
 49  PROCESS     FILEFAIL
 50  TESTFLOW    FMSGMISS                                                                    YES   NO
 51  LABEL       ERROR     *** LEAVE WITH REASON CODE SET
 52  LABEL       EXIT
 53  RETURN
 54  LABEL       FMSGMISS  ** FILE MSG NOT ON CURRENT CHAIN
 55  TESTFIELD   FILEFAIL  4                                                TSCAFRET          YES   NO   NO
 56  TESTFIELD   FILEFAIL  4                                                TSCAFRES          YES   NO   NO
 57  BRANCH      EXIT
 58  LABEL       EXERROR   ** ERROR GETTING TO UPDATE PANEL
 59  MOVEVAR               03
 60  USEREXIT              CALL BLGYAPSR - SET REASON         BLGYAPSR             NEXT     NO    NO
 61  BRANCH      EXIT
 62  LABEL       BADUPD    ** ERROR: UPDATE FAILED
 63  MOVEVAR               29
 64  USEREXIT              CALL BLGYAPSR - SET REASON         BLGYAPSR             NEXT     NO    NO
 65  BRANCH      CANCEL
 66  LABEL       CHECKOUT  ** ERROR: CHECKED OUT TO ANOTHER
 67  MOVEVAR               38
 68  USEREXIT              CALL BLGYAPSR - SET REASON         BLGYAPSR             NEXT     NO    NO
 69  BRANCH      CANCEL
 70  LABEL       FILEFAIL  ** ERROR: FILE OF RECORD FAILED
 71  MOVEVAR               03
 72  USEREXIT              CALL BLGYAPSR - SET REASON         BLGYAPSR             NEXT     NO    NO
 73  BRANCH      CANCEL
 74  LABEL       DISABLED  ** ERROR: SPECIAL PROC DISABLED
```

```
75  MOVEVAR             74
76  USEREXIT            CALL BLGYAPSR - SET REASON                BLGYAPSR            NEXT     NO    NO
77  BRANCH      CANCEL
78  LABEL       CNTLBLK   ** ERROR: BAD CONTROL BLOCK
79  MOVEVAR             82
80  USEREXIT            CALL BLGYAPSR - SET REASON                BLGYAPSR            NEXT     NO    NO
81  BRANCH      CANCEL
82  LABEL       NOTAUTH   ** ERROR: NOT AUTHORIZED
83  MOVEVAR             92
84  USEREXIT            CALL BLGYAPSR - SET REASON                BLGYAPSR            NEXT     NO    NO
85  BRANCH      CANCEL
86  LABEL       CANCEL    ** CANCEL UPDATE SESSION
87  ADDDATA             ;CANCEL                                                                      NO
88  PROCESS     EXIT
89  BRANCH      EXIT
90  LABEL       SAVEPNLN  *** SAVE SUMMARY PANEL NAME
91  SETFIELD                                                                  TSCAUFLD                YES
92  BRANCH      UPDATER
```

BLGAPI09–LLAPI Add Record Relation TSP for Panel Processing

This TSP performs the following steps:

1. Starts `;INIT,3,2` to go to the Management application PRIMARY OPTIONS MENU.
2. Issues `;UPDATE,1,5,2,update-rnid,` to update record and flow to the normal update summary panel.
3. Accesses panel BLG1AAUP and finds the control line containing the record type s-word from the update PIDT. If the TSP does not find a match, the TSP ends with an error. If the TSP finds a match, it saves the associated Update Summary panel in the TSCA variable area.

   If you are updating a record of your own type, you must build a PIDT with the correct record type s-word and update panel BLG1AAUP using the record type s-word for your record and its corresponding update summary panel.
4. Verifies that the current panel (update summary panel) is the same as the update summary panel read from BLG1AAUP. If the TSP does not find a match between the current panel name and the panel name read from BLG1AAUP, the TSP ends with an error. Possible causes of this error are specifying the wrong summary panel, or updating a record having a type other than that specified in the PIDT.
5. Verifies that the correct record is updated.
6. Adds data (user exit BLGYABPR).
7. Issues 9 to file the record. File time program exits and TSPs are started and notification is disabled.

This is TSP BLGAPI09.

```
TERMINAL SIMULATOR PANEL COMMON CONTROL FLOW DATA

LINE  FUNCTION     LABEL        LITERAL/TEST                 PANEL    FUNCTION   FIELD     WORD    APPLY  GET
NUM     NAME       NAME             DATA                     NAME       EXIT     NAME     OCCUR    NOT   VAR
---  ----------  --------  --------------------------------  --------  --------  --------  ------   ---   ---

  1  LABEL                 *** API ADD RECORD RELATIONS
  2  ADDDATA               ;INIT,3,2                                                                   NO
  3  PROCESS     EXERROR
  4  MOVEVAR               ;UPDATE,1,5,2,
  5  LABEL                 *** ADD THE RNID TO VAR. AREA
  6  USEREXIT              CALL BLGYAPBU                        BLGYAPBU            NEXT     NO    NO
  7  TESTFIELD   ERROR     4                                                TSCAFRET            NO    NO
  8  MOVEVAR                ,,
  9  ADDDATA                                                                                          YES
 10  PROCESS     EXERROR
 11  LABEL                 *** CHECK AUTH AND GET UPDATE
 12  LABEL                 *** SUMMARY PANEL NAME
 13  USEREXIT              CALL BLGYAPGP                        BLGYAPGP            NEXT     NO    NO
```

```
14  TESTFIELD   ERROR     4                                                TSCAFRET            NO    NO
15  TESTFIELD   BADUPD                                                     TSCACPNL            YES   YES
16  LABEL                 *** MAKE SURE UPDATE CORRECT REC
17  USEREXIT              CALL BLGYAPUP                          BLGYAPUP             NEXT    NO    NO
18  TESTFIELD   BADUPD    4                                                TSCAFRET            NO    NO
19  LABEL                 *** GET PIDT DATA AND ADD TO RAB
20  USEREXIT              CALL BLGYAPBR - ADD DATA               BLGYAPBR             NEXT    NO    NO
21  TESTFIELD   ERROR     4                                                TSCAFRET            NO    NO
22  LABEL                 *** FILE THE RECORD
23  ADDDATA               9                                                                    NO
24  PROCESS     EXERROR
25  TESTFLOW    EXERROR                                                                          YES   NO
26  LABEL       ERROR
27  LABEL       EXIT
28  RETURN
29  LABEL       BADUPD    *** SET COMMUNICATION AREA REASON
30  MOVEVAR               37
31  USEREXIT              CALL BLGYAPSR - SET REASON             BLGYAPSR             NEXT    NO    NO
32  BRANCH      EXIT
33  LABEL       EXERROR   *** SET COMMUNICATION AREA REASON
34  MOVEVAR               3
35  USEREXIT              CALL BLGYAPSR - SET REASON             BLGYAPSR             NEXT    NO    NO
36  BRANCH      EXIT
```

BLGAPI10–LLAPI Delete Record TSP

This TSP includes the following function for special processing.

- Delete a corrupted record based on the root VSAM key.

You can enable this function for processing by removing the DISABLED branch on line 9.

If you enable this special processing function, the TSP checks whether the application program has master or database administration authority. If the application does not have the required authority, the request fails, and a reason code is placed in PICAREAS.

You can change the authority required for special processing by changing the s-word index on line 42 (MOVEVAR). Refer to the section “S–Word Authorization Codes” in the *Tivoli Information Management for z/OS Panel Modification Facility Guide* for a list of s-word indexes.

This TSP performs the following steps:

1. Calls user exit BLGAPBU to obtain the RNID. If the root VSAM key was specified instead of the RNID, the key is first converted to the RNID, then the RNID is obtained.
2. If the RNID is obtained, the TSP:
   a. Processes the delete command.
   b. Starts TESTFLOW for record deleted message.
   c. Calls user exit BLGYAPSR on error to set the reason code.
   d. Returns.
3. If the RNID is not obtained, the TSP attempts to delete the record based on the root VSAM key.
   a. If the special processing disabling branch was not removed, exits.
   b. If the special processing branch was removed, checks for special processing authority. If the application does not have the required authority, puts a reason code in TSCAFRES and exits.
   c. If the application has the required authority, calls user exit BLGEXDEL to delete the record.
   d. Returns.

This is TSP BLGAPI10.

```
  TERMINAL SIMULATOR PANEL COMMON CONTROL FLOW DATA

 LINE  FUNCTION     LABEL         LITERAL/TEST                    PANEL    FUNCTION   FIELD     WORD    APPLY  GET  CASE-
 NUM     NAME       NAME             DATA                         NAME      EXIT     NAME      OCCUR    NOT   VAR  SENS
 ---  ----------  --------  --------------------------------  --------  --------  --------  ------  ---   ---  ---

   1  LABEL       BLGAPI10  *** API DELETE RECORD
   2  USEREXIT                                                          BLGYAPBU            NEXT     NO    NO
   3  TESTFIELD   RNIDVDA   4                                                     TSCAFRET            YES   NO   NO
   4  TESTFIELD   EXIT      0                                                     TSCAFRES            NO    NO   NO
   5  TESTFIELD   EXIT      16                                                    TSCAFRES            NO    NO   NO
   6  LABEL                 *** REMOVE FOLLOWING BRANCH TO
   7  LABEL                 *** ENABLE SPECIAL PROCESSING
   8  LABEL                 *** FOR DELETE RECORD
   9  BRANCH      DISABLED
  10  BRANCH      VSAMKEY
  11  LABEL       RNIDVDA   RNID ENTERED IN TSCAVDA
  12  ADDDATA               ;INIT,;DELETE R                                                       NO
  13  ADDDATA                                                                                     YES
  14  ADDDATA                ,2                                                                   NO
  15  PROCESS     BADPURGE
  16  TESTFLOW    EXIT                                                                    NO    NO
  17  LABEL       BADPURGE  CHECK WHY DELETE FAILED
  18  TESTFLOW    ERR03052                                                                NO    NO
  19  TESTFLOW    ERR03053                                                                NO    NO
  20  TESTFLOW    ERR03025                                                                NO    NO
  21  BRANCH      GENERROR
  22  LABEL       ERR03052  RECEIVED MESSAGE BLG03052
  23  MOVEVAR               12
  24  BRANCH      SETREASN
  25  LABEL       ERR03053  RECEIVED MESSAGE BLG03053
  26  MOVEVAR               38
  27  BRANCH      SETREASN
  28  LABEL       ERR03025  RECEIVED MESSAGE BLG03025
  29  MOVEVAR               10
  30  BRANCH      SETREASN
  31  LABEL       GENERROR  GENERAL DELETE ERROR REASON CODE
  32  MOVEVAR               03
  33  LABEL       SETREASN  SET API REASON CODE
  34  USEREXIT              CALL BLGYAPSR -SET DELETE REASON            BLGYAPSR            NEXT     NO    NO
  35  LABEL       EXIT      API DELETE TSP COMPLETE
  36  RETURN
  37  LABEL       VSAMKEY   *** USE VSAM ROOT KEY
  38  TESTFLOW    EXIT                                                                    YES   NO
  39  SETFIELD                                                                  TSCAUFLD            YES
  40  LABEL                 *** TO BEGIN AUTHORIZATION CHECK
  41  LABEL                 *** CALL USER EXIT BLGJAUTH
  42  MOVEVAR               0AA1
  43  USEREXIT                                                          BLGJAUTH            NEXT     NO    NO
  44  TESTFIELD   CNTLBLK   16                                                    TSCAFRET            NO    NO   NO
  45  TESTFIELD   NOTAUTH   0                                                     TSCAFRET            YES   NO   NO
  46  LABEL       AUTH      ** USER IS AUTHORIZED
  47  MOVEVAR                                                                   TSCAUFLD
  48  USEREXIT                                                          BLGEXDEL            NEXT     NO    NO
  49  TESTFIELD   EXIT      16                                                    TSCAFRET            YES   NO   NO
  50  LABEL       CNTLBLK   ** CONTROL BLOCK ERROR
  51  MOVEVAR               82
  52  BRANCH      SETREASN
  53  LABEL       DISABLED
  54  BRANCH      EXIT
  55  LABEL       NOTAUTH   ** SET COMMUNICATION AREA REASON
  56  MOVEVAR               92
  57  BRANCH      SETREASN
```


BLGAPIDI–LLAPI Router for Bypass Panel Processing

This panel is the controlling panel for the Application Program Interface (API) when Bypass Panel Processing is used. It performs the following:

1. Sets up the API environment and waits for a transaction.
2. Tests transaction code and links to transaction TSP.
3. On return, branches to API control user exit.
4. Quits/frees the API environment on error or T002 transaction.

This is TSP BLGAPIDI.

```
TERMINAL SIMULATOR PANEL COMMON CONTROL FLOW DATA

LINE  FUNCTION     LABEL        LITERAL/TEST                  PANEL    FUNCTION   FIELD     WORD    APPLY  GET  CASE-
NUM     NAME       NAME            DATA                       NAME      EXIT     NAME      OCCUR    NOT   VAR  SENS
---  ----------  --------  -------------------------------  --------  --------  --------  ------  ---   ---  ---

  1  LABEL       APIWAIT   *** WAIT FOR NEXT TRANSACTION
  2  USEREXIT              *** CALL API CONTROL USER EXIT             BLGYAPCP            NEXT     NO    NO
  3  LABEL                 TEST FOR TERMINATION
  4  TESTFIELD   LINKT002  T002                                                TSCAUFLD            NO    NO   NO
  5  LABEL                 TEST FOR CREATE TRANSACTION
  6  TESTFIELD   LINKT102  T102                                                TSCAUFLD            NO    NO   NO
  7  LABEL                 TEST FOR UPDATE TRANSACTION
  8  TESTFIELD   LINKT105  T105                                                TSCAUFLD            NO    NO   NO
  9  LABEL                 TEST FOR ADD RELATION XACTION
 10  TESTFIELD   LINKT109  T109                                                TSCAUFLD            NO    NO   NO
 11  LABEL                 TEST FOR DELETE TRANSACTION
 12  TESTFIELD   LINKT110  T110                                                TSCAUFLD            NO    NO   NO
 13  LABEL                 TEST FOR INVOKE USER TSP
 14  TESTFIELD   LINKT111  T111                                                TSCAUFLD            NO    NO   NO
 15  BRANCH      APIWAIT
 16  LABEL       LINKT102  LINK TO CREATE TSP
 17  LABEL       LINKT105  LINK TO UPDATE TSP
 18  LABEL       LINKT109  LINK TO ADD DATA TSP
 19  LINK                                                 BLGAPIPX
 20  BRANCH      APIWAIT
 21  LABEL       LINKT110  LINK TO DELETE TSP
 22  LINK                                                 BLGAPI10
 23  BRANCH      APIWAIT
 24  LABEL       LINKT111  INVOKE OR LINK TO USER TSP/TSX
 25  USEREXIT                                                   BLGYITSP            NEXT     NO    NO
 26  TESTFIELD   APIWAIT   0                                                   TSCAFRET            NO    NO   NO
 27  TESTFIELD   APIWAIT   4                                                   TSCAFRET            YES   NO   NO
 28  LABEL                 REPLACE THIS WITH LINK STATEMENT
 29  LABEL                 TO BRANCH TO HARDCODED TSP/TSX
 30  BRANCH      APIWAIT
 31  LABEL       INITERR   PRINT THE MESSAGES AND SCREEN
 32  PRINT
 33  LABEL       LINKT002
 34  ADDDATA               ;QUIT                                                                          NO
 35  PROCESS     ENDIT
 36  LABEL       ENDIT     LEAVE THE TSP
 37  RETURN
```

BLGAPIPX–LLAPI Bypass Panel Processing TSP

This TSP is the LLAPI Direct Interface TSP. It performs the following functions:

1. Call BLGYAPGP to see if history data or freeform text and check transaction authority.
2. If special processing is requested and when the disabling branch is removed, reset reason code to 0 and call user exit BLGJAUTH to check authorization.
3. ;INIT and call BLGYAPIS to set product.
4. Call BLGYAPBR to process transaction.
5. Call BLGYAPRF to file the record.

This is TSP BLGAPIPX.

```
TERMINAL SIMULATOR PANEL COMMON CONTROL FLOW DATA

LINE  FUNCTION     LABEL        LITERAL/TEST                  PANEL    FUNCTION   FIELD     WORD    APPLY  GET
NUM     NAME       NAME            DATA                       NAME      EXIT     NAME      OCCUR    NOT   VAR
---  ----------  --------  -------------------------------  --------  --------  --------  ------  ---   ---

  1  LABEL       BLGAPIPX  *** API PROCESS TRANSACTION
  2  LABEL                 *** INITIALIZE PRODUCT
  3  LABEL                 *** AND CHECK AUTHORIZATION
  4  USEREXIT              CALL BLGYAPGP - CHECK AUTHORITY            BLGYAPGP            NEXT     NO    NO
  5  LABEL                 *** BRANCH IF NO ERRORS AND NO
  6  LABEL                 *** SPECIAL PROCESSING REQUESTED
  7  TESTFIELD   RECORDP   0                                                   TSCAFRET            NO    NO
  8  TESTFIELD   EXIT      0                                                   TSCAFRES            NO    NO
```

```
 9  LABEL                *** REMOVE FOLLOWING BRANCH TO
10  LABEL                *** ENABLE SPECIAL PROCESSING
11  LABEL                *** FOR HISTORY DATA, FREEFORM
12  LABEL                *** TEXT
13  BRANCH      DISABLED
14  MOVEVAR              00
15  USEREXIT             CALL BLGYAPSR - RESET REASON              BLGYAPSR             NEXT     NO    NO
16  LABEL                *** TO BEGIN AUTHORIZATION CHECK
17  LABEL                *** CALL USER EXIT BLGJAUTH
18  MOVEVAR              0AA1
19  USEREXIT             CALL BLGJAUTH - CHECK AUTH                BLGJAUTH             NEXT     NO    NO
20  TESTFIELD   CNTLBLK  16                                                     TSCAFRET            NO    NO
21  TESTFIELD   NOTAUTH  0                                                      TSCAFRET            YES   NO
22  LABEL       RECORDP  *** PROCESS RECORD FUNCTION
23  LABEL                *** INITIALIZE PRODUCT
24  LABEL                *** AND BEGIN RECORD PROCESSING
25  ADDDATA              ;INIT                                                                                NO
26  PROCESS     ERROR
27  USEREXIT             CALL BLGYAPIS - SET PRODUCT               BLGYAPIS             NEXT     NO    NO
28  TESTFIELD   EXIT     0                                                      TSCAFRET            YES   NO
29  USEREXIT             CALL BLGYAPBR - PROCESS XACTION           BLGYAPBR             NEXT     NO    NO
30  USEREXIT             CALL BLGYAPRF - FILE RECORD               BLGYAPRF             NEXT     NO    NO
31  LABEL       EXIT
32  RETURN
33  LABEL       ERROR    ** ERROR: MESSAGE ISSUED
34  MOVEVAR              03
35  USEREXIT             CALL BLGYAPSR - SET REASON                BLGYAPSR             NEXT     NO    NO
36  BRANCH      EXIT
37  LABEL       DISABLED ** ERROR: SPECIAL PROC DISABLED
38  MOVEVAR              74
39  USEREXIT             CALL BLGYAPSR - SET REASON                BLGYAPSR             NEXT     NO    NO
40  BRANCH      EXIT
41  LABEL       CNTLBLK  ** ERROR: BAD CONTROL BLOCK
42  MOVEVAR              82
43  USEREXIT             CALL BLGYAPSR - SET REASON                BLGYAPSR             NEXT     NO    NO
44  BRANCH      EXIT
45  LABEL       NOTAUTH  ** ERROR: NOT AUTHORIZED
46  MOVEVAR              92
47  USEREXIT             CALL BLGYAPSR - SET REASON                BLGYAPSR             NEXT     NO    NO
48  BRANCH      EXIT
```

D

Record Process Panels

Note: The information in this section applies to panel processing mode.

The server is designed and implemented with terminal simulator panels (TSPs) that process transactions and control panels for process control. The TSPs call user exits that provide record and function processing.

Control panels provide panel dialog information used for record create and update. Unique control panels specify the dialog summary panels used during record create and update processing. The control panels are constructed so that each flow control line specifies a record s-word, and the true target panel name specifies the corresponding summary panel name. You can add control lines to each of these panels to accommodate additional user records.

Control panel BLG1AACP

Control panel BLG1AACP defines record type s-words and corresponding create summary panel names. It is used on API create processing and search processing to determine the record type for search hits and dynamic reference processing to determine the record type of the record being returned. You should define child record record type s-words before corresponding parent record record type s-words in BLG1AACP to ensure correct API processing.

Control panel BLG1AAUP

Control panel BLG1AAUP defines record type s-words and corresponding update summary panel names.

E

Sample Low-Level Application Program Interface

This example shows you what can be done with the Low-Level Application Program Interface (LLAPI) using the C language. Examine it to understand how you might code the interface to display data from an identified record, create a new record, or search for specific data in the Tivoli Information Management for z/OS database.

All sample parts are in the Tivoli Information Management for z/OS SAMPLIB (MVS data set SBLMSAMP). Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for information on the person to contact for information on high-level qualifiers of data sets at your site. See "Sample HLAPI/REXX Interface" on page 369 for more information on sample programs.

The sample code can provide an understanding of how you can do the following:

- ISPF starts the main routine SMPXMAIN.
- SMPXMAIN sets up the PICA, loads the LLAPI, and stores the addresses of both in ISPF variables (SMPCAADR and SMPSRADR).
- SMPXMAIN initializes the Tivoli Information Management for z/OS LLAPI (in asynchronous mode) and starts ISPF to SELECT menu panel SMPXPRIM.
- SMPXPRIM contains selections for three functions:
  - 0 - Display selected data about a specified problem record.
  - 1 - Create a new hardware component record.
  - 2 - Search for all non-CLOSED problems assigned to a specified department with assigned dates within a specified range.
- Each selection starts panels to collect the required data, then starts one of the C programs to interact with the LLAPI. The C program names are:
  - For selection 0 - SMPPRBRT (Problem Retrieve)
  - For selection 1 - SMPHWCCR (Hardware Component Create)
  - For selection 2 - SMPODPSR (OPEN Department Problem Search)
- The program starts the LLAPI to perform the function, then starts ISPF to display the data returned, if any, or a message.
- Upon return from the SMPXPRIM panel, SMPXMAIN gets control again, terminates the LLAPI, then exits back to ISPF.

Note!

The information contained in these sample programs has not been submitted to any formal IBM test and is distributed on an ″AS IS″ basis without warranty either expressed or implied. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any references made to an IBM program product are not intended to state or imply that only IBM's program product may be used; any functionally equivalent program may be used instead.

C Language Functions

The sample part BLMLLCS contains the following C language functions:

- SMPXMAIN - Sets up PICA, initializes and terminates LLAPI.
- SMPXCALI - Starts LLAPI, print return codes, and messages.
- SMPPRBRT - Retrieves data for a specified problem.
- SMPHWCCR - Creates a hardware component record.
- SMPODPSR - Searches for non-CLOSED problems assigned to a department during a date range.
- SMPPURGE - Deletes a record from the database.

These functions must be copied to separate source files to be compiled.

C Language Include File

The sample part BLMLLCU contains the following mappings:

- BLGPIAT - C structure mapping for PIAT control block
- BLGPICA - C structure mapping for PICA control block
- BLGPIDT - C structure mapping for PIDT control block
- BLGPIMB - C structure mapping for PIMB control block
- BLGPIPT - C structure mapping for PIPT control block
- BLGPIRT - C structure mapping for PIRT control block
- BLGPALT - C structure mapping for PALT control block

Note: The brackets used for arrays in the C source and include files are unprintable characters. They should be X'AD' (left bracket) and X'BD' (right bracket) on MVS for C/370™ to process them properly. This is a quirk in the way the C compiler works.

Panels

The ISPF panel definitions or the source for the customized PIDT referenced in SMPHWCCR have not been included in this document. (The program works using the standard hardware create PIDT.)

Using the C370 LLAPI Sample Programs

The C370 LLAPI sample programs require C370 R1.2 or higher to dynamically link to non-C language routines. To use these sample programs, perform the following steps:

1. Install Tivoli Information Management for z/OS and build a session-parameters member.
2. To verify that Tivoli Information Management for z/OS can be brought up using the session-parameters member built in step 1, start an interactive session of Tivoli Information Management for z/OS specifying that session-parameters member.
3. Copy the programs contained in the sample part BLMLLCS to a separate data set. These programs include the C language file BLMLLCU. You can separate the header files contained in BLMLLCU and include them in your programs.
4. Copy SMPXMAIN and SMPPURGE to other data sets, then modify them as follows:
   a. If you do not use BLGSES00, change BLGSES00 to the name of the session-parameters member you built in step 1.
   b. If you do not use MASTER, change MASTER to the name of the privilege class you want to use.
      - The privilege class you specify must have the authority to display and update problem records.
      - This privilege class must be in the database defined in the session-parameters member you specified in step 4a.
   c. Change all occurrences of SAMPID to the name of the application ID to use for the API session. The value you specify must be defined as an eligible user in the privilege class you use.
5. Build the ISPF panels for the programs.
6. Compile and link-edit the sample programs.
7. Verify that the required ISPF panels and Tivoli Information Management for z/OS code is available to the sample programs.
8. Run the sample program SMPXMAIN.

F

Sample High-Level Application Program Interface

These examples show what you can do with the High-Level Application Program Interface (HLAPI). Examine them to understand how you might code the interface to create, display, and update problem records, and create and display service records.

The C370 and PL/I HLAPI sample programs are in the Tivoli Information Management for z/OS SAMPLIB (MVS data set SBLMSAMP). Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for information on the person to contact for information on high-level qualifiers of data sets at your site. See "Sample HLAPI/REXX Interface" on page 369 for more information on sample programs.

Using the C370 HLAPI Sample Program

To use the C370 HLAPI sample programs, perform the following steps:

1. Install Tivoli Information Management for z/OS and build a session-parameters member.
2. To verify that Tivoli Information Management for z/OS can be brought up using the session-parameters member built in step 1, start an interactive session of Tivoli Information Management for z/OS specifying that session-parameters member.
3. Copy BLMHLCS to another data set, then modify it as follows:
   a. To specify the session-parameters member you built in step 1, change the value of `BLGSES00` on the statement `#define SESSMBR BLGSES00` to the name of your session-parameters member.
   b. To specify a privilege class, change the value of `MASTER` on the statement `#define PRIVCLAS MASTER` to the name of the privilege class you want to use.
      - The privilege class you specify must have the authority to create, display and update problem records.
      - This privilege class must be in the database defined in the session-parameters member you specified in step 3a.
   c. Change the value on the statement `#define APPLID` to the name of the application ID to use for the API session. The value you specify must be defined as an eligible user in the privilege class you use.
4. Copy BLMHLCJ to another data set.
5. Modify BLMHLCJ according to the instructions provided in the JCL.
6. Submit BLMHLJC to compile, link-edit, and run the sample program.

Using the PL/I HLAPI Sample Program

This sample program creates and retrieves a service record with record ID SAMPLE01. Delete this record before you run the sample program again. If you do not, the HLAPI transaction HL08 will fail because it cannot create a record with record ID SAMPLE01.

To use the PL/I HLAPI sample program in the Tivoli Information Management for z/OS SAMPLIB (MVS data set SBLMSAMP), perform the following steps:

1. Install Tivoli Information Management for z/OS and build a session-parameters member.
2. To verify that Tivoli Information Management for z/OS can be brought up using the session-parameters member built in step 1, start an interactive session of Tivoli Information Management for z/OS specifying that session-parameters member.
3. Copy BLMHLPS to another data set, then modify it as follows:
   a. To specify the session-parameters member you built in step 1, change all occurrences of `BLGSES00` to the name of your session-parameters member.
   b. To specify a privilege class, change all occurrences of `MASTER` to the name of the privilege class you want to use.
      - The privilege class you specify must have the authority to create and display service records.
      - This privilege class must be in the database defined in the session-parameters member you specified in step 3a.
   c. Change all occurrences of SAMPID to the name of the application ID to use for the API session. The value you specify must be defined as an eligible user in the privilege class you use.
4. Copy BLMHLPJ to another data set.
5. Modify BLMHLPJ according to the instructions provided in the JCL.
6. Submit BLMHLJC to compile, link-edit, and run the sample program.

G

Sample HLAPI/REXX Interface

This example shows what you can do with the High Level Application Program Interface for REXX (HLAPI/REXX). Examine it to understand how you might code the interface to update problem records. See "Output Data" on page 253 for information on the naming conventions of REXX variables used in this sample.

The HLAPI/REXX sample program is in the Tivoli Information Management for z/OS SAMPLIB (MVS data set SBLMSAMP). Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for information on the person to contact for information on high-level qualifiers of data sets at your site.

Using the HLAPI/REXX Sample Program

To use the sample program, perform the following steps:

1. Install Tivoli Information Management for z/OS and build a session-parameters member.
2. To verify that Tivoli Information Management for z/OS can be brought up using the session-parameters member built in step 1, start an interactive session of Tivoli Information Management for z/OS specifying that session-parameters member.
3. Copy BLMHLXS to another data set, then modify it as follows:
   a. If you do not use BLGSES00, change BLGSES00 to the name of the session-parameters member you built in step 1.
   b. If you do not use MASTER, change MASTER to the name of the privilege class you want to use. The privilege class you specify must have the authority to display and update problem records. This privilege class must be in the database defined in the session-parameters member you specified in step 3a.
   c. Change all occurrences of SAMPID to the name of the application ID to use for the API session. The value you specify must be defined as an eligible user in the privilege class you use.
4. Use Tivoli Information Management for z/OS to create one or more base problem records with the following:
   - A reporter name of `ADAMS/SAM`
   - A status of `OPEN`
   - A current priority greater than `1`
   - The text `TOM CARTER` in the description text.

   The program BLMHLXS will not find any matches if the database contains no records with this information.
5. Copy BLMHLXJ to another data set.

6. Modify BLMHLXJ according to the instructions provided in the JCL.
7. Submit BLMHLXJ to compile, link-edit, and run the sample program.

H

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 377 for more information about each book.

Typical Tasks

Table 91. 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* *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 91. 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* *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* *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 91. 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* *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*

I

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.

J

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

Numerics

A

B

C

D

E

F

G

H

M

N

O

P

R

S

T

U

V

W

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.

SC31-8737-00