TME 10 Information/Management

Problem Service User's Guide

Version 1.1

TME 10 Information/Management

Problem Service User's Guide

Version 1.1

First Edition (April 1998)

Copyright Notice

Trademarks

Notice

Contents

Preface

The *TME 10 Information/Management Problem Service User's Guide* describes the Problem Service component of Tivoli Management Environment Information/Management (TME 10 Information/Management).

Who Should Read This Guide

This book is intended for multiple audiences: the programmer who is developing an application that uses the Problem Service operations, and the end user of that application.

Prerequisite and Related Documents

The following is a list of TME 10 and other related publications:

TME 10 Framework User's Guide
Tivoli Management Platform Reference Manual
TCP/IP for MVS Customization and Administration Guide
OS/390 Security Server (RACF) Security Administrator's Guide

The following is a list of Information/Management publications:

Information/Management Problem, Change, and Configuration Management User's Guide
Information/Management Application Program Interface Guide
Information/Management Operation and Maintenance Reference
Information/Management Panel Modification Facility Guide and Reference

Information/Management Planning and Installation Guide and Reference
Information/Management Client Installation and User's Guide

What This Guide Contains

This book contains the following information:

- Chapter 1, Understanding the Problem Service Component

  Describes TME 10 Information/Management Problem Service, the application programming interface (API), and data mappings.

- Chapter 2, Problem Service Installation

  Describes installation planning considerations, requirements, and instructions.

- Chapter 3, Planning for Problem Service Configuration

  Describes how to plan for Problem Service configuration.

- Chapter 4, Customizing Your Problem Service Configuration File

  Contains the instructions for customizing the configuration file.

- Chapter 5, Completing Problem Service Configuration

  Describes the completion of Problem Service configuration, including instructions for updating the services file.

- Chapter 6, Running Problem Service

  Describes starting and stopping Problem Service.

- Appendix A, Appendix A, Application Programming Information

  This chapter is intended for the programmer who is writing an application to work with Problem Service.

- Appendix B, Appendix B, Customizing User Exit Routines for the Daemon

  Describes customizing user exit routines.

Typeface Conventions

This guide uses several typeface conventions for special terms and actions. These conventions have the following meaning:

Bold	Commands, keywords, file names, or other information that you must use literally appear in **bold**. Names of windows, dialogs, and other controls also appear in **bold**.
Italics	Variables and values that you must provide appear in *italics*.
Bold Italics	New terms appear in ***bold italics*** the first time they are used.
`Monospace`	Code examples appear in a `monospace` font.

Terminology

When the term ***MVS*** is used in this book, it means MVS/ESA or OS/390, unless otherwise stated.

When the term ***HLAPI client*** is used in this book, it means HLAPI client for AIX and HLAPI for Windows
NT client, unless otherwise stated.

For a complete glossary of TME 10 and related terms, refer to the *IBM Networking Glossary* located on the Web at http://www.networking.ibm.com/nsg/nsgmain.htm. This Web page enables you to search for terms and to download Portable Document Format (PDF) and PostScript glossary files.

Contacting Customer Support

We are very interested in hearing from you about your experience with the products and documentation in the Tivoli Management Environment. We welcome your suggestions for improvements.

If you encounter difficulties with any TME 10 product, please contact your customer support representative. To assist you, the TME 10

Framework includes the **wsupport** command. This command prompts you for problem information, which can be E-mailed to your support provider or saved to a text file. You can then print the saved file, and fax the resulting TME Problem Report form to your support provider. See the *TME 10 Framework Reference Manual* for additional information about the **wsupport** command.

If you have comments or suggestions about the TME 10 documentation, please send E-mail to USIB2HPD@VNET.IBM.COM.

1

Understanding the Problem Service Component

The Problem Service component provides distributed helpdesk applications with an interface to the Information/Management database on MVS so that both products can be part of the same distributed networking environment. Information/Management enables applications on remote environments to connect to the Information/Management system on MVS and to access the Information/Management database.

Problem Service consists of the following components:

- TME 10-related functions that install the Problem Service object and operations into the TME 10 database
- Daemon-related functions: the executable Problem Service daemon and user exits
- Sample files that provide coding examples for the Problem Service operations, configuration file, MVS tables, and high-level application program interface (HLAPI) profile
- Files needed to compile and link-edit an application that will use Problem Service operations

Conceptually, Problem Service provides:

- Session management with Information/Management
- A set of application programming interfaces (APIs) that enable you to interact with an Information/Management database

- A data mapping facility that enables you to define data manipulations

Problem Service Sessions

The sessions provided by Problem Service are determined by what you specify in the Problem Service configuration file. These sessions provide an application access through the HLAPI to an Information/Management database. There are three types of sessions available:

- Outbound
- Reverse
- Monitor

Multiple outbound sessions can be specified in the configuration file, but only one reverse session and one monitor session are permitted. Figure 1 on page 1-3 shows an overview of Problem Service sessions, including an interface with an application and an Information/Management database.

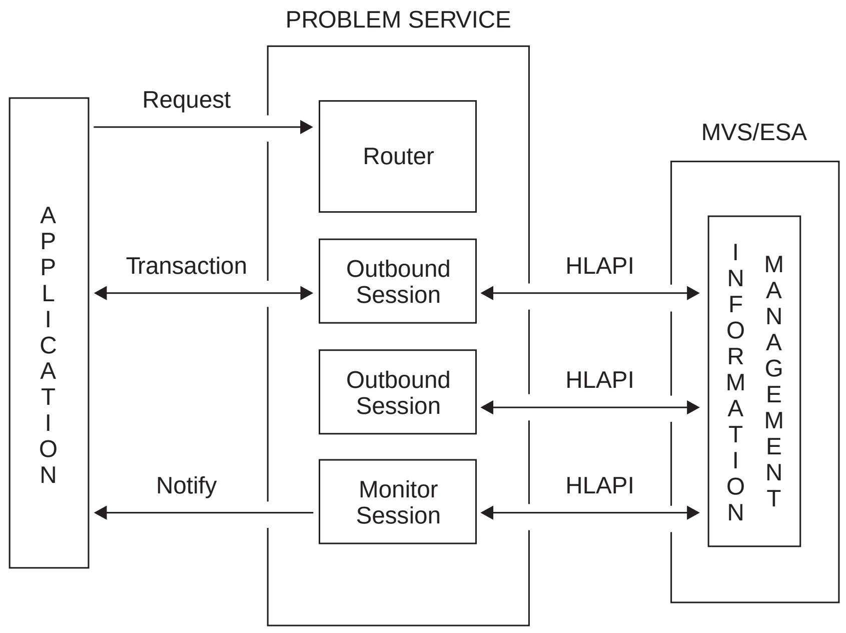

Figure 1. Problem Service Sessions Example

With this example, only three sessions were specified in the configuration file: one monitor session and two outbound sessions. A reverse session was not defined.

When the monitor session (polling daemon) detects a change in a database record that was assigned to or created by the application, it notifies the application.

The two outbound sessions are used for Problem Service operations initiated by the application. When the application requests an operation, Problem Service routes the request to the first available outbound session. A transaction interface is established and Problem Service performs the operation requested.

Problem Service Operations

Problem Service enables you to use your applications to share information in Information/Management database records. The following operations, using outbound sessions, enable you to exchange information between your applications and Information/Management databases:

- Checkin
- Checkout
- Delete
- Propagate
- Retrieve
- Search
- Transfer
- Update

The following are Problem Service control operations:

- Ping
- Shutdown

The following are Problem Service automated operations:

- Reverse assignment
- Monitor

The operations provided by Problem Service are affected by the availability of the network connection between the Problem Service daemons and Information/Management. If the network connection is unavailable, you cannot perform or complete some Problem Service operations. You can resume them when the connection becomes available.

Furthermore, the data fields handled by your application can differ in meaning and format from the data fields in Information/Management records. To ensure data integrity when these records are interchanged between your application and Information/Management, Problem Service provides customizable data mappings to map your application record fields to Information/Management record fields and vice versa.

To maintain data integrity, avoid multiple update access to the same record. Use the Problem Service record locking capability to ensure that your application keeps others from updating a record that you want to update.

A description of each Problem Service operation follows. Each description contains a prototype statement for the operation in the form of:

```
  Response   Operation   Input
```

where Input consists of

```
 (in datatype datavalue)
```

For example code fragments that show how to use the operations, see “Coding Examples for Problem Service Operations” on page A-3.

Unlocking a Record

The checkin operation unlocks a specified record in the Information/Management database by checking it in. The prototype statement for checkin is:

```
void checkin (in string rnid);
```

Where:

void	Indicates that there are no returned values.
rnid	Is a 1 to 8-character string that is the record number of the record to be checked in to Information/Management.

Locking a Record

The checkout operation locks a specified record in the Information/Management database by checking it out. The prototype statement for checkout is:

```
void checkout (in string rnid);
```

Where:

void	Indicates that there are no returned values.

rnid	Is a 1 to 8-character string that is the record number of the record to be checked out of Information/Management.

Deleting Records

The delete operation deletes a specified record in Information/Management. The prototype statement for delete is:

```
void delete (in string rnid);
```

Where:

void	Indicates that there are no returned values.
rnid	Is a 1 to 8-character string that is the record number of the record to be deleted from Information/Management.

Propagating Records

The propagate operation creates a copy of the record in the Information/Management database. When created in checked-out status, the Information/Management staff can only view the record to see any updates you make.

You can make this record a read-only copy by supplying the checkout indication data field or hardcoding the checkout indicator in the Problem Service configuration file data mappings for propagate.

The prototype statement for propagate is:

```
string propagate (in GWAttrList keyvaluepairs, in string recordtypevalue);
```

Where:

string	Is the Information/Management record number returned when the operation has successfully completed.
keyvaluepairs	Is a sequence of name value pairs consisting of name and data values.

recordtypevalue	Is the name of the record type that corresponds to the configuration file mapping section identified by the RecordTypeValue statement.

The record is assumed to exist if a record number (*rnid*) is passed as one of the name value pairs. The field that contains the *rnid* is determined by the ForeignIMRNIDField configuration file statement. In this case, the propagate operation attempts to update the existing Information/Management record.

Your application can pass only changed data or it can pass all record data. The InputJustChangedData statement in the Problem Service configuration file defines the selected behavior to Problem Service.

When you pass all data and use data mapping, Problem Service attempts to delete all NULL data fields from the Information/Management record. When you pass only changed values and you want to delete an Information/Management field, pass NULL as the data.

Retrieving Records

The retrieve operation retrieves a specific Information/Management record. The prototype statement for retrieve is:

```
GWAttrList retrieve (in string rnid, in string recordtypevalue);
```

Where:

GWAttrList	Is an Information/Management record in a sequence of name value pairs returned in response to the retrieve operation.
rnid	Is a 1 to 8-character string that is the record number of the record to be retrieved from Information/Management.
recordtypevalue	Is the name of the record type that corresponds to the configuration file mapping section identified by the RecordTypeValue statement.

Searching for a Record

The search operation returns a list of Information/Management records that match the specified search criteria. The prototype statement for search is:

```
SearchResultList search (in GWAttrList keyvaluepairs, in string recordtypevalue);
```

Where:

SearchResultList	Is a sequence of elements that match the search criteria of the search operation. Each element contains the *rnid* and a data value, which is typically the data abstract, for each Information/Management record. The AssociatedDataField statement in the Problem Service configuration file determines the Information/Management data field that is returned in the name value pair. The number of matches is limited to the value of the SearchHits statement in the Problem Service configuration file.
keyvaluepairs	Is the sequence of name value pairs consisting of name and data values which specify the criteria to be used for the search operation.
recordtypevalue	Is the name of the record type that corresponds to the configuration file mapping section identified with the RecordTypeValue statement.

Transferring Records

The transfer operation creates a record in the Information/Management database that can be viewed and updated by the Information/Management staff. The prototype statement for transfer is:

```
string transfer (in GWAttrList keyvaluepairs, in string recordtypevalue);
```

Where:

string	Is the Information/Management record number returned when the operation has successfully completed.
keyvaluepairs	Is a sequence of name value pairs consisting of name and data values. The *rnid* can be specified as one of the name/value pairs or it can be assigned by Information/Management. When Information/Management assigns the *rnid*, it is always an 8-character numeric string.
recordtypevalue	Is the name of the record type that corresponds to the configuration file mapping section identified by the RecordTypeValue statement.

Updating Records

The update operation updates the specified Information/Management record. The prototype statement for update is:

```
void update (in GWAttrList keyvaluepairs, in string recordtypevalue);
```

Where:

void	Indicates there are no returned values.
keyvaluepairs	Is the sequence of name value pairs that consist of name and data values. One of the name value pairs must contain an *rnid*. The field that contains the *rnid* is determined by the ForeignIMRNIDField configuration file statement.
recordtypevalue	Is the name of the record type that corresponds to the configuration file mapping section identified by the RecordTypeValue statement.

Your application can pass only changed data or it can pass all record data. The InputJustChangedData statement in the Problem Service configuration file defines the selected behavior to Problem Service.

If you pass all data and use data mapping, Problem Service attempts to delete all NULL data fields from the Information/Management record. When you pass only changed values and you want to delete an Information/Management field, pass NULL as the data.

Control Operations

Here are the descriptions for the control operations (ping and shutdown) provided by Problem Service.

Pinging for Status

Ping is used by your application to obtain the status of the Problem Service daemons. The prototype statement for ping is:

```
PingResult ping ();
```

The operation returns *PingResult*, which indicates the running status of Problem Service. If Problem Service is not running, an exception is thrown.

Shutting Down Problem Service

Shutdown closes the Problem Service and its associated child daemons. This is a hard shutdown, and does not ensure that all transactions are completed. The prototype statement for shutdown is:

```
void shutdown ();
```

No values are returned.

Automated Operations

The remaining operations provided by Problem Service are automated operations. You tailor the Problem Service configuration file to customize the behavior of the operations. There can only be one session specified for each automated operation in the configuration file: one reverse session and one monitor session.

Reverse Assignment Operation

Reverse assignment allows records created in Information/Management to be used and assigned in the local database. You might want to transfer the responsibility of specific Information/Management records to your application's staff or have your staff share the responsibility for those records with the Information/Management staff. Periodically, reverse assignment automatically queries the Information/Management database and identifies records that meet specific search criteria.

The record is still owned by Information/Management and the record can be updated by the Information/Management staff. When you update a reverse assigned record, the record should be checked-out and retrieved from the Information/Management database before it is updated.

Reverse assigned records can be monitored for changes. See “Monitor Operation” on page 1-12 for more information on monitoring.

Tailoring: You tailor the Problem Service configuration file to customize the behavior of this operation by specifying:

- The time interval between each activation of the reverse assignment operation.
- The search criteria to use when querying the Information/Management database.
- The executable to be invoked when search matches are found.

Activating: The reverse assignment automated operation is activated at the end of each time interval. If the network connection to Information/Management is not available at activation time, the reverse assignment operation is not performed for that activation.

If the network connection is available at activation time, reverse assignment queries the Information/Management database according to the specified search criteria. When it queries the database, it considers all records except those that have already been transferred or reverse assigned by your application's system. The number of search matches is limited to the SearchHits configuration file value.

Record Processing: For each record that is identified in the Information/Management database, the executable identified by the RACallApp statement is run on the system identified by the ForeignHost statement. These statements and their values can be found in your configuration file.

The record number, record type, and the reverse assignment GatewayID (R*GatewayID*) are passed to the executable, which performs the processing of the Information/Management record. This processing could include:

1. Checking out the record.
2. Retrieving the record.
3. Performing application-specific actions (for example, creating a record in the application's database).
4. Updating the Information/Management record to add the gateway ID to mark the record as being reverse assigned by this Problem Service. This enables the monitor operation on the record.

   Note: This step prevents the reverse assignment operation from repeatedly processing the same record.
5. Checking in the record.

Monitor Operation

Problem Service provides an automated monitor operation that periodically queries the Information/Management database to identify records that have been transferred or reverse assigned by your application and that have been recently updated in Information/Management.

Tailoring: You tailor the Problem Service configuration file to specify:

- The time interval between each activation of the monitor operation.
- The executable to be invoked when search hits are found.

Activating: The monitor operation is activated at the end of each time interval. If the network connection to Information/Management is not available at activation time, the monitor operation is not performed for that activation.

The monitor operation affects only the records that have been identified as being reverse assigned or transferred by your application's system. The records processed by Problem Service contain a gateway identifier, specified in the Problem Service configuration file, that associates them with your application's system that is running Problem Service.

Note: This assumes that the transfer data mappings include the Problem Service identifier and that you updated reverse assigned records with the Problem Service identifier.

The monitor operation selects only records that were last changed by a different user. All records last changed by your application through a Problem Service instance will be marked as being changed last by the gateway identifier value.

You must update the Information/Management record or the monitor will always find the same record on the next monitor iteration. If you do not have data to change, you can use a field returned by the retrieve operation as the data to use for the update.

Record Processing: For each record that the monitor finds in the Information/Management database that matches the search criteria, the executable identified by the MonCallApp statement is run on the system identified by the ForeignHost statement. These statements and their values are in your configuration file.

The record number and record type are passed to the executable. The number of search matches is limited to the SearchHits configuration file value. The executable processes the Information/Management record, which could include:

1. Checking out the record.
2. Retrieving the record.
3. Performing an application-specific function (for example, updating a record in the application's database).
4. Updating the Information/Management record to mark the record as being last altered by this instance of Problem Service.

   Note: This step prevents the monitor operation from repeatedly processing the same record.

5. Checking in the record.

Problem Service Data Mappings

When a propagate or transfer operation copies a record to Information/Management, it uses the data you pass to it as input for the Information/Management record fields. Correspondingly, when you retrieve an Information/Management record, it uses the data in the Information/Management record fields as input. However, the record fields in your application's data might differ in number, name, meaning, and format from the Information/Management record fields. You must define how these record contents are mapped from one database to the other.

The Problem Service data mappings consist of mapping rules, specified in the Problem Service configuration file, that the Problem Service operations apply when manipulating records. You can customize the data mappings for each operation to specify:

- Which of your data fields and Information/Management record fields are processed for various operations.
- The association between your data fields and Information/Management record fields that have similar meaning.
- The conversion mechanism to use to transform the field data from your format to the Information/Management record format and from the Information/Management record format to your application's data format.

You can disable all data mapping by specifying `PerformDataMapping=no` in the configuration file. When data mapping is disabled, you work with only Information/Management field names and Information/Management format data.

2

Problem Service Installation

This chapter explains how to plan for and install Problem Service.

Follow these steps to install and configure Problem Service:

1. Ensure that the prerequisite software is installed, configured, and operational on your workstation and on the MVS host.
2. Install Problem Service on your workstation.

   Note: The Problem Service daemon (AIX) or the Windows NT service must be installed on a Tivoli Management Region (TMR) server.
3. Configure Problem Service on your workstation (see "Customizing Your Problem Service Configuration File" on page 4-1).

   Note: Work with the Information/Management system administrator to complete the Problem Service configuration.

Planning for Problem Service Installation

The following sections describe how Problem Service fits into the Information/Management environment and provides you with an overview of the Information/Management setup that is necessary for operating Problem Service.

Information/Management Environment

To perform its operations, Problem Service uses the Information/Management HLAPI to remotely access the Information/Management database records from the workstation environment.

The Information/Management HLAPI that is part of the MVS Information/Management product is a transaction-based application programming interface (API). It enables applications in remote environments to establish HLAPI working sessions with the host Information/Management system to manipulate database records.

For an overview of the Information/Management product and features, refer to the *Information/Management Planning and Installation Guide and Reference*.

HLAPI Client

In the workstation environment, Information/Management provides an HLAPI client that establishes communication links with the Information/Management system on MVS. This feature of Information/Management is made up of two interfaces: a requester and a client. Both interfaces must be installed and configured in the workstation environment.

Requester Interface

The requester interface provides the communication link to the Information/Management system on MVS. Either TCP/IP or advanced program-to-program communication (APPC) is used, depending on the platform used.

Client Interface

The client interface must reside on the Problem Service workstation. It provides Problem Service with the interface to the Information/Management HLAPI.

For more information on setting up the HLAPI client/server environment, refer to the *Information/Management Client Installation and User's Guide*.

Installation Requirements

To install and configure Problem Service on your workstation, you need the hardware, storage, and software listed in the following sections.

Hardware Requirements

The Problem Service option has no unique hardware requirements.

Disk Space Requirements

In addition to your workstation's other requirements, the free disk space required for Problem Service is shown in Table 1.

Table 1. Free Disk Space Requirements

Problem Service	**Disk Space**
AIX	12 MB
Windows NT	7 MB

Software Requirements

The installation and operation of Problem Service requires software on both MVS and the workstation.

MVS Host

To use Problem Service, you must have Information/Management Version 6.3 or later installed and operational on an MVS host. Refer to the *Information/Managment Planning and Installation Guide and Reference* for more information.

The Information/Management HLAPI and either a remote environment server (RES) or multiclient remote environment server (MRES) must be set up on the Information/Management system on MVS that owns the database that your Problem Service must access.

Refer to the *Information/Management Planning and Installation Guide and Reference* and to the *Information/Management Client Installation* *and User's Guide* for information on the Information/Management environment setup for remote HLAPI applications.

AIX Workstation

The following software is required for the installation, operation, and maintenance of Problem Service:

- AIX Version 4 Release 1
- Information/Management, Version 6.3 or later, HLAPI Client for AIX feature and associated software prerequisites
- Tivoli Management Environment 3.1

  Note: When writing an application to use Problem Service, you also need the TME 10 Application Development Environment.

For instructions on how to install and configure the HLAPI Client for AIX, refer to the *Information/Management Client Installation and User's Guide* Install and configure both the requester and the client interface components on the AIX workstation.

Note: The client interface must reside on the same workstation where Problem Service is to be installed.

After you install and configure the HLAPI Client for AIX, record the name and location of the database profile you configure. You need this information for configuring Problem Service. If you did not change the name and location of this file, you can find it in the **/usr/lpp/idbhlapi/examples** directory. The default name is **idbdb.pro**. You can also customize and use the example HLAPI profile **gateway.prf** supplied with Problem Service.

Windows NT Workstation

The following software is required for the installation, operation, and maintenance of Problem Service:

- Windows NT 4.0
- Information/Management, Version 6.3 or later, HLAPI for Windows NT Client feature and associated software prerequisites

- Tivoli Management Environment 3.1

  Note: When writing an application to use Problem Service, you also need TME 10 Application Development Environment.

For instructions on how to install and configure the HLAPI for Windows NT Client, refer to the *Information/Management Client Installation and User's Guide*. Install and configure both the requester and the client interface components on the Windows NT workstation where Problem Service is to be installed.

After you install and configure the HLAPI for Windows NT Client, record the name and location of the database profile you configure. You need this information for configuring Problem Service. If you did not change the name and location of this file, you can find it in the **c:\infoapi\sample** directory. The default name is **database.pro**. You can also customize and use the example HLAPI profile **gateway.prf** supplied with Problem Service.

Installing Problem Service

Before installing Problem Service, ensure that the HLAPI client can communicate with Information/Management on MVS.

Some of the commands that you are requested to use in this section require you to be in the TME 10 environment. For AIX, entering the TME 10 environment is typically done by invoking the **setup_env.sh** shell script in the session being used. For Windows NT, entering the TME 10 environment is typically done by running the **setup_env.cmd** command file.

Use the TME 10 desktop to install Problem Service on your TMR server by clicking on **Desktop** → **Install** → **Install Product.** To ensure that the installation completed successfully, enter the following command from the TME 10 environment to obtain the object ID (OID) of the object:

```
wlookup -r InfoMgtGW -a
```

The Info_GW instance is displayed, labeled with the OID. Your application uses the OID of the Info_GW instance to invoke Problem Service operations.

National Language Support (NLS) for Messages

After installing Problem Service, the message catalog resides in the **/msg_cat/C** directory under the TME 10 install tree. For AIX this directory needs to be part of your NLSPATH.

AIX Workstations

For catalog files to reside in your NLSPATH, issue the **export LANG=C** command after you have entered the TME 10 environment. When the **bl*.cat** files reside in the NLSPATH, the messages issued by the daemon resolve correctly.

Windows NT Workstations

LOCPATH points to the directories used to convert data between different code sets. For example, you can set the LOCPATH variable by specifying:

```
LOCPATH=C:\INFOAPI\LOCALE
```

The LOCPATH variable must be set in the system environment variables on the TMR server where Problem Service is installed. You must be logged on as an administrator to update system environment variables.

REGSRV2 Program (Windows NT Only)

Problem Service is automatically registered as a Windows NT service as part of the installation process. If it should fail for any reason, or if you need to register or unregister Problem Service, information on running REGSRV2 is provided here. You can change the drive letter and directory path if necessary.

To install Problem Service (gw_nxd) as a Windows NT service, use the REGSRV2 program. The format is:

```
REGSRV2 x app_name
```

Where:

x
: Is I to install the service or D to delete the service.

app_name
: Is the full application path name of the program, when the Windows NT service is being installed. If the full application path name contains a blank, enclose it in quotes. For example:

```
REGSRV2 I "C:\Program Files\GW_NXD.EXE"
```

The *app_name* parameter is not needed when deleting the Windows NT service. For example:

```
REGSRV2 D
```

Note: You must be logged on as an administrator to install the Windows NT service.

3

Planning for Problem Service Configuration

This chapter provides a general overview of some basic Information/Management concepts and the process for configuring Problem Service.

Basic HLAPI Concepts

The Problem Service configuration requires you to be familiar with the following HLAPI-related concepts:

- HLAPI transactions
- The input and control Parameter Data Block (PDB), used for HLAPI transactions
- HLAPI data views that consist of either:
  - Program interface data tables (PIDTs) and program interface pattern tables (PIPTs)
  - Data model records
- Program interface alias table (PALT)

This section provides introductory information about these concepts. For a more detailed description, refer to the *Information/Management Application Program Interface Guide*.

HLAPI Transactions

HLAPI is a transaction-based application program interface. Problem Service uses HLAPI transactions to establish sessions with the Information/Management system and to manipulate its database records.

In particular, to perform the Problem Service operations as described in "Problem Service Operations" on page 1-4, Problem Service uses the HLAPI transactions:

- checkin
- checkout
- create
- inquiry
- retrieve
- update

For example, the Problem Service propagate and transfer operations use the HLAPI create transaction to create a corresponding record in the Information/Management database. The propagate operation also uses an implicit checkout on the create transaction to lock the record in the Information/Management database. This protects the propagated record from being updated by the Information/Management staff or other users and allows only the application that propagated the record to update it.

HLAPI PDBs

The HLAPI PDBs contain parameters that control the operating characteristics of the HLAPI sessions that Problem Service uses to connect to the Information/Management system and to manipulate the records in the database. Problem Service uses these PDBs for the HLAPI transactions.

HLAPI Data Views

HLAPI uses data views to define the Information/Management record fields that an application can access using the HLAPI transactions. These views can be PIDTs and PIPTs or data model (data view, data attribute, and data validation) records.

In an Information/Management database that is not customized, PIDTs and associated PIPTs define the base Information/Management records. You can customize these PIDTs and PIPTs to define additional user-defined fields in the Information/Management records that you want your application to manipulate. You could also build data model records that define your fields.

Each HLAPI transaction has a separate PIDT that is also associated with a specific Information/Management record type. For example, the HLAPI create transaction has a separate PIDT that defines the fields that can be used when creating a record in the Information/Management database.

If you use data model records, the data view record can apply to multiple record transactions. For example, you can use the same data view for HLAPI create, update, or retrieve transactions.

Structured and Prefix Word Indexes

The PIDT or data attributes identify specific fields within an Information/Management record using Information/Management ***structured word*** (s-word) and ***prefix word*** (p-word) indexes. An s-word index is represented by an S, followed by 4 hexadecimal characters. An example of an s-word index is `S0B5C`. A p-word index is represented by a P, followed by 4 hexadecimal characters. An example of a p-word index is `P028A`.

P-Words

You should be familiar with p-words and their uses. A p-word consists of a keyword that performs searches on fields in Information/Management database records. It can be associated to one or several s-word indexes. A p-word can be up to 6 characters long and must include the slash (/) or underscore (_) character as the last character. An example of a p-word is `AUTH/`.

To specify Information/Management fields that HLAPI recognizes in the Problem Service configuration file, you can use indexes, alias names, and in some cases, p-words. For simplicity, the sample Problem Service configuration file uses s-word indexes instead of alias names.

HLAPI PALTs

The PALT enables applications to specify alias names for PIDTs, p-words, p-word indexes, and s-word indexes. In a PALT, you can specify default values for the Information/Management record fields. That is, you can specify default response data values for Information/Management record fields for which your application does not provide a response value.

You can define an alias name for a field, and use the alias name instead of using an s-word or a p-word index to identify the field. An alias name can as long as 32 characters. For example, you could define and use an alias name of `STATUS` instead of using the s-word index of `S0BEE`.

Problem Service Configuration Process

Problem Service configuration is a multistep process that must be done to enable Problem Service operations. This process also involves customizing the Information/Management system.

Note: Work with the Information/Management administrator to complete the Problem Service configuration process.

The following sample files are provided by Problem Service for the configuration process:

- A sample configuration file that you customize for your application's environment to use Problem Service.
- A sample set of Information/Management-specific files to assist you in customizing the tables that define the Information/Management record fields. Problem Service accesses these record fields when performing HLAPI transactions on the records.

Sample Configuration File

A configuration file used by Problem Service controls various aspects of how Problem Service operates in your application's system and environment. It is a root-owned file and access to it is usually restricted to your application's administrator.

The sample configuration file **blmygc.cfg** is provided with Problem Service, but it must be customized before use. The **blmygc.cfg** file is made up of a series of statements that are grouped into three main parts:

1. Information/Management HLAPI session information
2. Problem Service general settings
3. Data mappings

This file is located in the **$INST_DIR/$INTERP/InfoMgt/InfoGateway** directory.

Customizing Statements

To customize the sample configuration file statements, edit the file and modify the statements as required. Knowledge of the Information/Management HLAPI and how it is used by remote applications is a prerequisite for editing and modifying the statements. Refer to the *Information/Management Application Program Interface Guide* for a description of the HLAPI. References to other Information/Management manuals are indicated when necessary.

The sample file has already been partially customized for you with suggested values that suit any application's system. You only need to customize a few statements in it that are related to your specific application's environment.

Customize the required values in the sample file that have been pre-filled with question mark (?) characters. After you are familiar with this file, you can customize any of the remaining statements.

General Syntax Rules

The general syntax to follow when modifying the configuration file is the following:

- Each statement in the configuration file must end with a semicolon. If the statement spans more than one line, only the last line in the statement should end with a semicolon.
- Comments can appear on any line in the file but must be preceded with double slash (//) characters.

Process Steps

To configure Problem Service, perform the following tasks:

1. Customize the Problem Service sample configuration file:
    a. Customize the Information/Management HLAPI information that Problem Service uses when establishing sessions with Information/Management. See “Customizing the HLAPI Session Information” on page 4-1 for instructions.
    b. Customize the general Problem Service settings required for Problem Service operations. See “Customizing Problem Service General Settings” on page 4-10 for instructions.
    c. Customize the data mappings that Problem Service applies when transferring records back and forth between your applications and the Information/Management database. See “Customizing Problem Service Data Mappings” on page 4-20 for instructions.
2. Prepare the HLAPI data views on MVS. See “Preparing the HLAPI Data Views on MVS” on page 5-1 for instructions.
3. Update the Services file. See “Updating the Services File” on page 5-8 for instructions.

You can customize the Problem Service configuration file at any time to modify existing settings. If your applications and Problem Service are running:

1. Stop your applications.

2. Stop Problem Service.
3. Modify the Problem Service configuration file.
4. Start Problem Service for the new configuration settings to take effect.
5. Start your applications

4

Customizing Your Problem Service Configuration File

This chapter explains how to plan for and configure the Problem Service option.

Customizing the HLAPI Session Information

The configuration file contains a series of HLAPI-related statements that correspond to the HLAPI control PDB parameters. Problem Service uses these PDB parameters when establishing HLAPI sessions with Information/Management and when performing transactions on records in the Information/Management database. These statement values are used for all sessions started between Problem Service and Information/Management.

The configuration file also contains HLAPI session statements that determine the number and the characteristics of the sessions that Problem Service will establish.

HLAPI-Related Statements

Figure 2 on page 4-2 shows the HLAPI-related statements in the sample configuration file.

```
// HLAPI-Related Statements

SessionMember="BLGSES??";
PrivilegeClass="?????";
TableCount=5;
APIMsgOption="B";
HLIMsgOption="B";
SpoolInterval=0;
TimeoutInterval=120;
DatabaseID="5";
DefaultOption="NONE";
DefaultDataStorageSize=1024;
SeparatorCharacter=",";
BypassPanel="NO";
ReplaceFreeFormText=yes;
```

Figure 2. HLAPI-Related Statements in the Sample Configuration File

Customize the statements that appear in bold characters. These statements are required and are specific to your application's system. Ask your Information/Management program administrator to provide you with these required values.

Table 2 on page 4-3 gives you a brief description of the HLAPI-related statements. As each of these statements corresponds to a specific parameter of the HLAPI control PDB, the correspondence with these parameters is also included in the table.

For more detailed descriptions of these parameters and the values they can assume, refer to the *Information/Management Application Program Interface Guide*.

Table 2 (Page 1 of 5). HLAPI Related-Statements for Sessions

Statement / **PDB Parameter**	**Description**
SessionMember SESSION_MEMBER	This is a required statement. It specifies the name of the Information/Management session-parameters member to be used for sessions with the HLAPI client. The name can be up to 8 characters long, beginning with the character string **BLGSES**. It must match the name of the session member that the Information/Management administrator defines in MVS.
PrivilegeClass PRIVILEGE_CLASS	This is a required statement. It specifies the level of authority granted to users for performing certain operations on the Information/Management database. Your Problem Service application is considered an Information/Management user. This value must match the name of the privilege class that the Information/Management administrator assigns to your Problem Service application. The value can be 1 to 8 characters long.
TableCount TABLE_COUNT	This is an optional statement. It specifies the number of Information/Management tables that can be in storage during the session. The value ranges from 0 to 256. The default value is 0. The suggested value for Problem Service is 5.

Table 2 (Page 2 of 5). HLAPI Related-Statements for Sessions

Statement	
PDB Parameter	**Description**
APIMsgOption APIMSG_OPTION	This is an optional statement. It specifies the destination of the low level application program interface (LLAPI) logging messages. The valid values are P, C, or B. P indicates that LLAPI writes messages to the APIPRINT data set, and C indicates that the LLAPI messages will be put in message PDBs. The default value is B, which indicates that LLAPI will perform both P and C. This statement is used only if the SpoolInterval statement is specified and has a non-zero value.
HLIMsgOption HLIMSG_OPTION	This is an optional statement. It specifies the destination of the HLAPI logging messages. The valid values are P, C, or B. P indicates that HLAPI writes messages to the HLAPILOG data set, and C indicates that HLAPI puts the messages in the message PDBs. The default value is B, which indicates that HLAPI will perform both P and C. This statement is used only if the SpoolInterval statement is specified and has a non-zero value.

Table 2 (Page 3 of 5). HLAPI Related-Statements for Sessions

Statement / **PDB Parameter**	**Description**
SpoolInterval SPOOL_INTERVAL	This is an optional statement. It specifies the time interval, in minutes, for HLAPI and LLAPI logging in MVS. The value ranges from 0 to 1440. The default value is 0, meaning that messages are not logged. In this case, the values in HLIMsgOption and APIMsgOption are ignored.
TimeoutInterval TIMEOUT_INTERVAL	This is an optional statement. It specifies the number of seconds that a database transaction can run before a timeout causes HLAPI to terminate the session. The value ranges from 0 to 300, but if you specify a value between 0 and 45, the interval is set to 45 seconds. The default value is 300 seconds. The suggested value is 120.
DatabaseID DATABASE_ID	This is an optional statement. It specifies the name or ID number of the database that your application accesses during the session. For Information/Management records that can be created or updated, the database ID is 5; do not change this value. The default value is 5.

Table 2 (Page 4 of 5). HLAPI Related-Statements for Sessions

Statement PDB Parameter	Description
DefaultOption DEFAULT_OPTION	This is an optional statement. It specifies which Information/Management record fields are candidates for default data response processing by HLAPI during the session. The valid values are: ▪ REQUIRED ▪ ALL ▪ NONE The default value is NONE.
DefaultDataStorageSize DEFAULT_DATA_STORAGE_SIZE	This is an optional statement. It specifies the additional storage, in bytes, that HLAPI uses to hold the default response data. The default value is 1024.
SeparatorCharacter SEPARATOR_CHARACTER	This is an optional statement. It specifies the separator character that HLAPI uses to process response data. This character is used by HLAPI to separate the data items in a record field, if the field contains a list of data items. The default value is the comma.
BypassPanel BYPASS_PANEL_PROCESSING	This is an optional statement. It is used globally for all Information/Management API sessions that this Problem Service starts. Only used on the API initialization transaction; applies for all of the transactions the gateway performs. The default value is NO.

Table 2 (Page 5 of 5). HLAPI Related-Statements for Sessions

Statement / PDB Parameter	Description
ReplaceFreeformText REPLACE_TEXT_DATA	This is an optional statement. It is used globally for the Information/Management API sessions that this Problem Service starts. It is used on an update transaction or a propagate transaction when propagating a record that has already been propagated (results in an update of the Information/Management record). If it has a value of yes, the update uses the REPLACE_TEXT_DATA Information/Management HLAPI control PDB with a value of YES. This indicates that new freeform text being supplied with the update replaces existing freeform text of the same type in the record being updated. Problem Service does not specify the REPLACE_FREEFORM_TEXT PDB if ReplaceFreeformText has a value of NO. The default value is YES.

HLAPI Session Statements

The other HLAPI statements that you must customize in the configuration file are the HLAPI session statements. These statements determine:

- The number of sessions that Problem Service will establish with the Information/Management HLAPI to perform Problem Service operations on the Information/Management database.
- The characteristics of the physical connection to the MVS host where Information/Management resides.
- The type of Problem Service operation for which Problem Service will use each session

The number of sessions is determined by the number of Session statements specified in the configuration file.

The following example shows the session statements as they appear in the Problem Service sample configuration file. Customize the keywords in bold characters.

```
// Session statements
Session Transaction=outbound
   DatabaseProfile="/usr/lpp/idbhlapi/examples/idbdb.pro"
   UserID="?????"   Password="?????";
Session Transaction=outbound
   DatabaseProfile="/usr/lpp/idbhlapi/examples/idbdb.pro"
   UserID="?????"   Password="?????";
Session Transaction=reverse
   DatabaseProfile="/usr/lpp/idbhlapi/examples/idbdb.pro"
   UserID="?????"   Password="?????";
Session Transaction=monitor
   DatabaseProfile="/usr/lpp/idbhlapi/examples/idbdb.pro"
   UserID="?????"   Password="?????";
```

Each session statement has four keywords: Transaction, DatabaseProfile, UserID, and Password.

Transaction
This keyword specifies the type of Problem Service operations for which the session will be used.

The values are:

monitor	Indicates monitor operations.
outbound	Indicates that the session will be used for all operations except monitor and reverse assignment.
reverse	Indicates reverse assignment operations.

You can specify only one session statement for each of the monitor and reverse assignment operations. You can specify up to 30 session statements for outbound operations. A separate daemon is started for each session.

You should specify at least one session for each type of transaction to obtain full Problem Service functionality. Otherwise, the Problem Service operation corresponding to an unspecified transaction will not be performed. For example, when you do not specify the reverse transaction in any of the session statements, Problem Service does not perform reverse assignment operations.

DatabaseProfile
The name of the database profile that you configured for the HLAPI client.

The database profile determines the connection characteristics for the session. If you are using the default database profile from the HLAPI client installation, use:

- For AIX:

  ```
  /usr/lpp/idbhlapi/examples/idbdb.pro
  ```

- For Windows NT:

  ```
  x:\infoapi\sample\database.pro
  ```

  where *x* is the drive on which the HLAPI client was installed.

If you are using a different database profile, you must specify the path and name of that file.

Note: Omit the path specification if the database profile resides in the current working directory for Problem Service.

Refer to the *Information/Management Client Installation and User's Guide* for more information on the database profile usage and specification.

UserID
A user ID assigned for the MVS logon for the session.

This user ID is assigned to you by the MVS administrator and it must have the necessary authorizations to access the Information/Management database. UserID corresponds to the SECURITY_ID parameter in the HLAPI control PDB used for establishing the session.

Password
The password assigned for the MVS logon for the session.

This parameter corresponds to the PASSWORD parameter in the HLAPI control PDB used for establishing the session.

The sessions you customize can share a single physical connection to the MVS system or can use separate physical connections. If you specify the same values for UserID, Password, and DatabaseProfile in all the session statements, the sessions will share a single physical connection. Only one HLAPI transaction can be performed at a time over each physical connection.

If your organization intends to use Problem Service operations extensively, you can improve performance by forcing separate physical connections for some sessions. One way of doing this is by specifying different UserID and Password values for those sessions.

Refer to the *Information/Management Client Installation and User's Guide* for information about other ways to force separate physical connections.

Customizing Problem Service General Settings

The statements in the second part of the configuration file define general settings that control the operating characteristics of Problem Service. Figure 3 on page 4-11 provides partially customized statements as they appear in the sample configuration file. You must customize the statements that appear in bold characters. If any of the remaining customized statements do not suit your application's system requirements, modify them.

```
// Problem Service General Settings
GatewayID="?????";
ReverseAssignInterval=60;
MonitorInterval=60;
InfoGatewayService="infogateway";
ForeignHost="?????";
RACallAPP="rassign";
MonCallApp="moninfo";
UnconditionalShutdownWait=10;
GatewayIDField="S1260";
GatewayIDPrefix="GWID/";
ConvertToUppercase=yes;
MaximumHits=1000;
MaxTextRetrieveLines=20;
IMPatternValidation=yes;
IMRNIDField="S0CCF";
IMCheckoutField="S14EF";
AssociatedDataField="S0E0F';
RetrieveBeforeUpdate=no;
SessionRetryInterval=10;
SessionRetryLimit=10;
InputJustChangedData=no;
PerformDataMapping=yes;
```

Figure 3. Problem Service General Settings Statements in the Sample Configuration File

The following list gives a description of the statements in this part of the configuration file and explains how to customize them.

GatewayID (Required)

Defines the unique identifier for a Problem Service instance. It can be 1 to 8 characters long.

GatewayID corresponds to the APPLICATION_ID parameter in the HLAPI control PDBs. The ID you specify must be an eligible user ID defined in the same privilege class that you use in the **PrivilegeClass** statement in the configuration file.

Problem Service uses the GatewayID value for two main purposes:

- To identify itself as an eligible Information/Management user, so that it can perform database record transactions during the HLAPI sessions with Information/Management.

The same GatewayID value is used for all the HLAPI sessions it establishes with Information/Management.

- To identify all the Information/Management records that it processes so that they can later be associated to this particular Problem Service application. The GatewayID value is stored in a field present in all Information/Management records it processes. It also stores a prefix character, along with the GatewayID value, to indicate the operation performed on each of these records:

Prefix	**Definition**
P	Propagated
T	Transferred
R	Reverse assigned

When you customize the data mappings in the configuration file, specify the GatewayID value and prefix character in the propagate and transfer data mappings. For the records that your application reverse assigns, it must update the Information/Management record's GatewayIDField with a reverse assignment gateway ID (R*GatewayID*). This keeps your application from being notified of the same Information/Management records over and over again.

The Problem Service monitor operation uses this GatewayID value to identify all the records transferred and reverse assigned by this particular Problem Service.

Note: In an Information/Management environment where more than one Problem Service or instance is present, you must make sure that the GatewayID is unique for each Problem Service. Otherwise, Problem Service operations will produce unpredictable results.

It is recommended that you use the GatewayID value as the checkout indicator value when customizing the data mappings for the propagate operation. This causes an automatic check-out of the records that are propagated by your application and enables you to update these records.

Note: After you are assigned a GatewayID value, you should not change it. However, if such a need arises, in addition to changing the value in the GatewayID statement, you must change all the occurrences of the GatewayID value in the rest of the Problem Service configuration file. Otherwise, unpredictable results will occur.

Furthermore, if you change the GatewayID value, all the records in Information/Management that were previously processed by Problem Service will contain the old GatewayID value.

ReverseAssignInterval (Optional)
Specifies the time interval, in minutes, that Problem Service waits between each activation of the reverse assignment operation.

The value ranges from 0 to 999999.

The default value is 60 minutes.

MonitorInterval (Optional)
Specifies the time interval, in minutes, that Problem Service waits between each activation of the monitor operation.

The value ranges from 0 to 999999.

The default value is 60 minutes.

InfoGatewayService (Required)
Specifies the service name of the TCP/IP port used by Problem Service.

The sample configuration file provides the InfoGatewayService service name, which must be *infogateway*. Define this service name in the services file, and assign it an available TCP/IP port number.

ForeignHost (Required)
Specifies the name of the host that holds the executables specified by the RACallApp and MonCallApp statements in the configuration file.

To determine the host name for AIX, issue the **hostname** AIX command from the AIX command line on the system that holds the executables.

To determine the hostname for a Windows NT system, on the Windows NT desktop select **My Computer → Control Panel→ Network→ Protocols→ TCP/IP Protocol→ Properties→ DNS**.

RACallApp (Optional)
Specifies the name of the executable to be invoked whenever the reverse assignment operation finds a search match in the Information/Management database. The host name of the system where the executable resides is specified by the ForeignHost statement.

This executable should be written to accept three parameters: the Information/Management record ID, its record type, and the reverse assignment gateway ID to be used to update the Information/Management record when the application accepts the record.

MonCallApp (Optional)
Specifies the name of the executable to be invoked whenever the monitor operation finds a record in the Information/Management database that belongs to this application's gateway ID and has been updated by Information/Management or another application. The host name of the system where the executable resides is specified by the ForeignHost statement.

This executable should be written to accept two parameters: the Information/Management record ID and its record type.

UnconditionalShutdownWait (Optional)
Specifies the number of seconds that Problem Service waits before sending unconditional shut down signals to terminate unclosed subprocesses, after having previously attempted to shut down those processes.

The value range is 5 to 300 seconds.

The default value is 10 seconds.

GatewayIDField (Required)
Specifies the index or alias name of an Information/Management record field that will hold the GatewayID value. Problem Service needs such a field to store the GatewayID value and prefix character for all the records it propagates and transfers. For reverse assigns, you must store the GatewayID value in this field.

Problem Service defines a new field in Information/Management records to hold the GatewayID value. The sample configuration file defines this new field with s-word index S1260. The Problem Service installation provides the HLAPI PIDTs and PIPTs that define this new field in Information/Management records. The HLAPI create, retrieve, and update PIDTs or data views for records must contain this field. See “Preparing the HLAPI Data Views on MVS” on page 5-1 for instructions on how to use the PIDTs and PIPTs provided.

You can use the suggested and defined Information/Management field or use another field (either an existing one or another user-defined field) to hold the GatewayID value. If you choose to use another Information/Management field, customize the create, update, and retrieve PIDTs and PIPTs or your data model records in the Information/Management system on MVS to include this field. See “Preparing the HLAPI Data Views on MVS” on page 5-1 for information on customizing PIDTs and PIPTs.

GatewayIDPrefix (Required)
Specifies the p-word that is associated with the Information/Management GatewayID field. It is used for searching the Information/Management database for the records processed by Problem Service.

A p-word can contain up to 6 characters, where the last character must be a slash (/) or an underscore (_).

The suggested value provided in the sample configuration file is `GWID/`.

ConvertToUppercase (Optional)

Specifies whether or not to convert to uppercase characters all field data, except for freeform text data, that Problem Service stores in the Information/Management database.

The possible values are:

yes Indicates that the data will be converted to uppercase characters.

no Indicates that the data will not be converted.

The default value is yes.

MaximumHits (Optional)

Corresponds to the NUMBER_OF_HITS parameter of the HLAPI control PDB. It is used for the HLAPI inquiry transaction on Information/Management records. It specifies the maximum number of matches to be returned from a search. It applies to the Problem Service monitor, reverse assignment, and search operations.

The value ranges from 0 to 9999. The value 0 is treated as if the MaximumHits statement has not been specified.

The default value is 500. The suggested value for Problem Service is 1000.

MaxTextRetrieveLines (Optional)

Corresponds to the TEXT_UNITS parameter of the HLAPI control PDB. It specifies the maximum number of lines that can be retrieved for each freeform text field of an Information/Management record.

The value ranges from 1 to 9999.

The default value is 1000. The suggested value for Problem Service is 20.

IMPatternValidation (Optional)

Specifies whether or not HLAPI should perform pattern validation on the input data fields for the records that Problem Service creates or updates in the Information/Management database.

The possible values are:

yes Indicates that pattern validation will be performed.

no Indicates that pattern validation will not be performed.

The default value is yes.

IMRNIDField (Required)
Specifies the index or alias name of the Information/Management record field that holds the record number identifier for the Information/Management record.

For Information/Management records, the s-word index for the default RNID field is S0CCF.

IMCheckoutField (Required)
Specifies the index or alias name of the Information/Management record field that is used as the indicator that the record is locked or checked out in Information/Management.

The Information/Management records already contain a field that Information/Management uses to indicate that records are checked out. The s-word index of this field is S14EF, which you must specify for this statement.

The PIDTs and PIPTs provided by the Problem Service installation contain this field. HLAPI created PIDTs or data views for database records must contain this field so that Problem Service can use this field for propagated records. See "Preparing the HLAPI Data Views on MVS" on page 5-1 for instructions on how to use the PIDTs and PIPTs provided.

Note: Specify this field in the propagate data mappings, in the third part of the Problem Service configuration file, so that Problem Service automatically checks out propagated records.

AssociatedDataField (Required)
Specifies the index or alias of the Information/Management field that is to be returned for matches on the search operation.

RetrieveBeforeUpdate (Optional)

Specifies whether Problem Service needs to perform special processing before updating a record in the Information/Management database. This statement is applicable when you are using record fields that contain an Information/Management List Processor list of data items. It also affects how Problem Service handles the deletion of null fields when updating the Information/Management record.

The possible values are:

yes Indicates that Problem Service will first retrieve the record fields from the Information/Management database and then update them.

Note: The retrieve is not performed if data mapping is disabled.

no Indicates that Problem Service will update the record fields directly, without retrieving them.

The default value is yes. The suggested value for Problem Service is no.

If the record is retrieved before the update, Problem Service can verify the fields that are already null and does not need to perform extra delete processing to delete their contents. If the record is not retrieved, Problem Service attempts to delete the field contents even if the fields are already empty. This has an impact on performance if the record contains a large number of empty fields.

When updating an Information/Management List Processor list with a shortened list, you can specify **yes** to ensure the Information/Management field is updated correctly. If you specify **no**, you must explicitly delete unwanted list entries by using the HLAPI separator character.

SessionRetryInterval (Required)

Specifies the time interval, in minutes, that Problem Service will delay between attempts to start a session that has stopped for some reason other than being shut down.

The maximum value for the interval is 30 minutes.

SessionRetryLimit (Required)
Specifies the number of times that Problem Service will attempt to restart a stopped session.

The maximum value for the number of retries is 10. A value of 0 directs Problem Service not to attempt session restarts.

Note: It is recommended that you initially set this value to zero (0). After Problem Service is installed and configured, a higher value can be set.

InputJustChangedData (Optional)
Specifies whether the caller will provide all data for a record or just the changed (delta) data. This is important when performing an Information/Management update (updating or propagating) to an existing record.

yes	Problem Service attempts to delete from the Information/Management record null fields passed by the caller. Special list processor field processing is not performed. The caller must ensure that existing entries in a list to be updated are deleted if necessary. This is important when passing a parameter list that has fewer entries than the current list it is updating.
no	Problem Service attempts to delete Information/Management fields that are defined in your data mappings, but were not passed by your application or hardcoded in the data mappings.

The default value is no.

PerformDataMapping (Optional)
The possible values are:

yes	The data mapping function of the gateway is used.

no	Data mapping is not performed; all data is in Information/Management format. If updating an Information/Management record, optional record retrieval and null field deletion is not performed, unless you are only passing changed (delta) data.

The default value is yes.

Customizing Problem Service Data Mappings

The main purpose of the third part of the configuration file is to define the data mappings that will be applied during Problem Service operations and Information/Management transactions that share data between Information/Management and your application's records. If you are an end user of Problem Service, refer to your application's documentation for help in mapping records to Information/Management.

This part of the configuration file also contains statements that define the data required by the Information/Management HLAPI transactions as well as statements that enable you to customize the search criteria for the reverse assignment operation. You must complete the customization of this part of the configuration file before you can use Problem Service.

The data mappings in the sample configuration file have been customized to best fit a sample application's requirements and to find the best match between the Information/Management database fields and the application's record fields. The mappings are based on uncustomized Information/Management records. If they have been customized, the data mappings need to be adjusted accordingly. You can disable Problem Service data mapping by specifying:

```
PerformDataMapping=no
```

Mapping Your Application and Information/Management Records

Fields within mapped records that are contained in the Information/Management record and your application's record will be mapped. Remember that the mappings in the sample configuration file are based on uncustomized Information/Management records. You can configure the mapping to:

- Alter a mapped-to Information/Management record field.
- Add a new mapped-to Information/Management record field.
- Remove a mapped-to Information/Management record field.

Information/Management list processor and multiple response fields are supported. List processor fields contain from 1 to 19274 entries. An example is a name field that allows a first and last name separated by a blank. Most Information/Management fields are single response fields.

HLAPI requires that lists and multiple responses for a field be entered as separate strings with each response separated by a separator character (default is a comma). The data for these fields must be passed to Problem Service containing these separator characters or must be manipulated by data mappings to include them. For example, the data can be separated by blanks when passed to Problem Service and the mappings can convert these blanks into the separator character.

The default mappings supplied in the sample configuration file do not contain list processor fields. The only Information/Management fields that are multiple response fields are those customized by the user.

See “Supported Data Conversions” on page B-1 for information on converting data and for a list of the data conversions supplied with Problem Service.

Setting Up the Data Mapping Rules

Fields can be mapped between your application's records and Information/Management records. Using mapping statements in the configuration file, data is mapped to produce a collection of data that has Information/Management data keys and Information/Management format.

For transactions, such as propagate and transfer, that put a record into the Information/Management database, a mapping statement with a hardcoded gateway ID (with prefix) should be included. The prefix indicates whether the Information/Management record has been propagated or transferred.

Prefix	**Definition**
P	Record has been propagated.
T	Record has been transferred.

In the default mappings, the gateway ID is identified by the Information/Management s-word index S1260.

Table 3 shows the uncustomized Information/Management record fields to which the sample application record fields are mapped by the Problem Service samples.

Table 3 (Page 1 of 3). Uncustomized Information/Management and Sample Application Record Fields and Their Attributes

Uncustomized Information/Management Field			**Sample Application Field**	
Name	**Index**	**Attributes**	**Name**	**Attributes**
Assignee department	S0B9C	11 alphanumeric characters including #, @, $, &, or /	Organization	30 characters, varchar
Assignee name	S0B5A	1–15 alphanumeric characters including #, @, $, &, or /	Assignee	90 characters, varchar
Current priority	S0BE7	1–2 numeric	Priority	1 digit (1, 2, 3, 4, 5) or None (required)

Table 3 (Page 2 of 3). Uncustomized Information/Management and Sample Application Record Fields and Their Attributes

Uncustomized Information/Management Field			Sample Application Field	
Name	**Index**	**Attributes**	**Name**	**Attributes**
Date/Time opened	S0C3E or S0C74	Date: external date format; Time: external time format	StartDate	MM/DD/YY(YY) hh:mm:ss (a|p)
Description abstract	S0E0F	1–45 freeform (string)	(not mapped)	
Description text	S0E01	Freeform text	Description	240 characters, freeform text
Gateway ID	S1260	1–8 alphanumeric characters including #, @, $, &, or /	(not mapped)	
Problem status	S0BEE	INITIAL OPEN CLOSED	Status	Approved Closed Open Working Pending Rejected Complete
Problem type	S0C09	1–8 alphameric characters including #, @, $, &, or /	TroubleCode (hierarchy classifying the problem)	Length is 40, data can include '.'
Record ID	S0CCF	1–8 alphanumeric characters including #, @, $, &, or /	rnid	30 characters, varchar
Reported by	S0B59	1–15 alphanumeric characters including #, @, $, &, or /	Originator	90 characters, varchar
Status text	S0E02	Freeform text	Detail	1000 characters, freeform text
Tracked by	S0B5C	1–15 alphanumeric characters including #, @, $, &, or /	Modifier	90 characters (first, middle, last)

Table 3 (Page 3 of 3). Uncustomized Information/Management and Sample Application Record Fields and Their Attributes

Uncustomized Information/Management Field			**Sample Application Field**	
Name	**Index**	**Attributes**	**Name**	**Attributes**
Vendor PMR number	S0F52	8 numeric	TicketNum	8 numeric

Mapping Fields to Problem Service Operations

Not all fields are mapped for each operation being performed. In the configuration file, specify the fields that are to be mapped for each operation by using transactions statements. They enable fields to be mapped to one or more Problem Service operations. Different fields can be mapped to different operations.

Data mapping statements specified before the first occurrence of a transactions statement apply to all operations. Otherwise, the data mapping statements apply only to the operations specified by the most recent transactions statement. For example:

```
Transactions=propagate,transfer,update;
 S0B5A(15)<<translate(Assignee,", ","//");
```

In this example, the data mapping statement following the transactions statement applies to the Problem Service operations: propagate, transfer, and update.

Multiple transactions statements can be specified. When multiple transactions statements are specified with the same keyword, the mapping definitions for each occurrence are grouped together for that operation; the last occurrence does not override previous occurrences.

Understanding the Syntax of Data Mapping Statements

Data mapping statements identify the data to be mapped, how to transform the data, and where to put the data. The direction of the mapping is indicated by double less than symbols (<<) or double greater than symbols (>>):

<<	Specifies that the mapping applies to data flowing from your application's database to the Information/Management database. The target of the mapping is the operand to the left of the <<.
>>	Specifies that the mapping applies to data flowing from the Information/Management database to your application's database. The target of the mapping is the operand to the right of the >>.

The mapping target operand is a field name, suffixed by a length enclosed in parentheses. Field names appearing to the left of the << or >> are assumed to be Information/Management field names, and field names appearing to the right of the << or >> are assumed to be your application's field names. An Information/Management field name can be one of the following:

- An alias name as defined in a program alias table (PALT).
- An S followed by an s-word index as defined in the PIDT field PIDTSYMB.
- A P followed by a prefix index as defined in the PIDT field PIDTSYMB.

The source for the mapping is a combination of literal strings, field names, and user exit specifications. Data associated with source field names, and data returned from user exit calls, is substituted by the mapping facility. User exits are specified as user exit name, followed by a comma delimited argument list enclosed in parentheses. Each of these arguments can be a combination of integers, literal strings, field names, and user exit specifications. See “Supported Data Conversions” on page B-1 for information on data conversions and user exits.

The syntax for a mapping into an Information/Management field is as follows:

```
IMFieldName(length) << literal_string                 ;
                       subroutineName(parm1, parm2, ...)
                       your_application's_FieldName
                       [combination of the above]
```

Where `parm`*n* can be:

```
  integer
  literal_string
  your_application's_FieldName
  subroutineName(parm1, parm2, ...)
```

The following is an example of a mapping into your application's field:

```
literal_string      >> your_application's_FieldName(length);
subroutineName(parm1, parm2, ...)
IMFieldName
[combination of the above]
```

Where `parm`*n* can be:

```
  integer
  literal_string
  IMFieldName
  subroutineName(parm1, parm2, ...)
```

Syntax Examples: The following are some examples of data mappings in both directions:

- `S0E0F<<"This is a description abstract.";`

  For appropriate transaction types (for example, transfer), the Information/Management field defined in the PIDT as S0E0F is assigned the value `This is a description abstract.`

- `S0E02>>detail(1000);`

  For appropriate transaction types (for example, retrieve), the data in the Information/Management field defined in the PIDT as S0E02 is assigned to your application's field named detail. The target field is truncated or padded with blanks to 1000 bytes.

- `"@"S0B59>>Originator;`

For appropriate transaction types (for example, retrieve), an @ is added as a prefix to the data in the Information/Management field defined in the PIDT as S0B59. The result of this operation is then assigned to your application's field named Originator.

Changing the Data Mapping Rules

You can change data mappings rules by adding, modifying, or deleting mapping statements in the configuration file.

Adding a Field: When a new field is added, a new mapping statement must be added to the configuration file if the field is to be shared with Information/Management. All necessary conversions must be specified here. The Information/Management PIDTs and PIPTs must be rebuilt or data model records modified to allow Problem Service to process the new field in Information/Management.

Changing a Field: If a field in your application's record is changed, it might be necessary to change the mapping statements by:

- Converting it in a different way
- Mapping it into a different Information/Management field
- Changing the operating characteristics of Problem Service

Information/Management PIDTs and PIPTs might have to be rebuilt to allow the Information/Management HLAPI to correctly validate data sent by Problem Service. Keep the Problem Service and Information/Management data models as similar as possible to minimize the data conversions needed to share data between the two databases.

Removing a Field: When a field in your application's record is removed, check the configuration file to ensure that there are no mapping statements mapping Information/Management fields into the removed field. If there are, these mapping statements need to be removed from the configuration file.

Sample Configuration File Descriptions

In the sample configuration file, values have been included that you need not change for a basic configuration. It assumes that Information/Management records have not been customized. Values that you **must** provide are clearly indicated by bold characters or question mark (?) characters in the following descriptions.

Data mappings should be defined for all fields that are to be shared between your application and Information/Management records.

Defining Specific Record Types

The part of the configuration file shown in Figure 4 on page 4-29 contains statements needed by the Information/Management HLAPI. The statement descriptions are:

```
//****************************************************************************
//
// Define information to use for specific record types including data
// required by the Information/Management HLAPI and data mappings for
// data sharing between the local application and Information/Management.
//
//****************************************************************************

//****************************************************************************
//
// Record Type:  Helpdeskapp
//
// Define information for a Helpdeskapp record.  Helpdeskapp
// records map to Information/Management records.
//
//****************************************************************************
RecordType

    // ----------------------------------------------------------------------
    // String that identifies the record type.
    // ----------------------------------------------------------------------
    RecordTypeValue="Helpdeskapp"
    // ----------------------------------------------------------------------
    // Key of field in helpdesk record that contains the
    // Information/Management RNID.
    // ----------------------------------------------------------------------
    ForeignIMRNIDField="rnid"

    // ----------------------------------------------------------------------
    // Key of field (PIDTSYMB value or alias name) that contains the identifier
    // of the helpdesk record - only used by Reverse Assignment and Monitor.
    // ----------------------------------------------------------------------
    IMForeignRNIDField="S0F52"
```

Figure 4. Sample Configuration File Record Type Statements

RecordTypeValue="Helpdeskapp"
The string that identifies the record type. Identify the record type as the appropriate record type for your application.

ForeignIMRNIDField="rnid"
The name of the field in your application record that contains the Information/Management identifier.

IMForeignRNIDField="S0F52"
The Information/Management field (PIDTSYMB value or alias name) that contains the identifier of your application record. This is only used by the reverse assignment and monitor operations.

Another value that can be defined, but which has not been included in the sample configuration file, is:

AliasTable
This defines the value for the Information/Management HLAPI control PDB ALIAS_TABLE. When this statement is specified, the ALIAS_TABLE control PDB is specified for all Information/Management HLAPI create, update, retrieve, and inquiry transactions performed for this record type.

Defining API PIDT Names

The required Information/Management API PIDT names are specified by the following configuration file statements:

```
CreateDataView="P:BLMYGPC"
UpdateDataView="P:BLMYGPU"
DisplayDataView="P:BLMYGPR"
SearchDataView="P:BLGYPRI";
```

They specify the set of Information/Management PIDTs or data view record names to be used with the record type field value (see the RecordTypeValue keyword). These statements can be specified more than one time in the configuration file, but are locally mutually inclusive (all must appear wherever one appears). These statements must immediately follow a RecordTypeValue statement.

The data value for these PIDT statements must begin with a P:, or a D:. P: indicates that the rest of the value is a PIDT name (for example, P:BLGYPRC). D: indicates that the rest of the value is a data view record ID.

CreateDataView="P:BLMYGPC"
This is the name of the Problem Service sample PIDT that creates Information/Management records.

UpdateDataView="P:BLMYGPU"
This is the name of the Problem Service sample PIDT that updates Information/Management records.

DisplayDataView="P:BLMYGPR"
This is the name of the Problem Service sample PIDT that retrieves Information/Management records.

SearchDataView="P:BLGYPRI"
This is the name of the Problem Service sample inquiry PIDT table used in Information/Management.

Defining Freeform Text Fields

The following sample configuration file statement defines the Information/Management fields that are freeform text together with their associated line widths:

```
 IMText Width=60 Fields="S0E01", "S0E02";
```

In the example, the Information/Management fields S0E01 and S0E02 are freeform text fields with a line width of 60 characters.

The application's record text is broken up into segments using the width value with each segment becoming a line of text in the Information/Management record. The Problem Service transfer, propagate, and update operations use the value specified for width to break up the application's input data into Information/Management freeform text lines of this width. The retrieve operation uses the maximum width for the record type as the width to retrieve, and this value then becomes the value of the Information/Management HLAPI control PDB TEXT_UNITS.

Information/Management freeform text lines are truncated or padded with blanks to the width specified.

Defining the ReverseArguments Statement

One or more structured or freeform search arguments can be specified for the reverse assignment operation. If no arguments are specified, reverse assignment will not be performed. The following is a sample configuration file ReverseArguments statement:

```
ReverseArguments S0B5A="??????";
```

For example, when **SMITH** replaces **??????** in the sample, it becomes:

```
ReverseArguments S0B5A="SMITH";
```

This means that all records with the field S0B5A containing the value SMITH will be reverse assigned. This is an example of a structured search argument. If a search argument is specified by itself, without an associated field name, it is a freeform search.

If more than one search argument is specified, all conditions must be satisfied for the record to be reverse assigned. Arguments can include a Boolean operator as the first character (valid operators are those allowed by the Information/Management HLAPI for freeform search arguments). In the following example, all records for which the S0B5A field contains SMITH and not reported by JONES will be reverse assigned.

```
ReverseArguments S0B5A="SMITH" "êPERS/JONES";
```

Reverse assignment can act against records transferred by other gateways, but not against records transferred by this gateway. When a transferred record is reverse assigned, the originating Problem Service stops monitoring it for changes. Reverse assignment does not act against records reverse assigned by another gateway. It checks the value in the GatewayID field. It also ignores records that have been checked out.

ReverseArguments constructs the total search argument using the following information:

- ReverseArguments structured arguments
- Predefined arguments (for example, not transferred by this gateway)
- ReverseArguments freeform arguments

Mapping Records from Your Application to Information/Management

The mappings as shown in Figure 5 are defined for the Problem Service propagate, transfer, and update operations as specified in the transactions statement. Your application's data is mapped to Information/Management data through the use of user exits. For information, on these user exits, refer to “Specifying User Exits for Conversions” on page B-4.

```
Transactions=propagate,transfer,update;
 S0B5A(15)<<translate(Assignee,", ","//");
 S0B9C(11)<<translate(Organization,", ","//");
 S0B5C(15)<<translate(Modifier,", ","//");
 S0BE7<<change(Priority,"None","");
 S0C3E<<toIMDate(words(StartDate,1,1));
 S0C74<<toIMTime(words(StartDate,2));
 S0BEE<<translateWord(Status,"Approved","OPEN",
                             "Pending","OPEN",
                             "Working","OPEN",
                             "Complete","CLOSED",
                             "Rejected","CLOSED");
 S0C09(8)<<translate(TroubleCode,".","/");
 S0B59(15)<<nullDefault(translate(Originator,", ","//"),
                             "Helpdeskapp") ;
```

Figure 5. Propagate, Transfer, and Update Operations Mappings

Where:

S0B5A(15)<<translate(Assignee,", ","//");
All commas and spaces found within the value in the application's **Assignee** field are converted to slashes and the value is truncated to 15 characters.

S0B9C(11)<<translate(Organization,", ","//");
All commas and spaces found within the value in the application's **Organization** field are converted to slashes and the value is truncated to 11 characters.

S0B5C(15)<<translate(Modifier,", ","//");
All commas and spaces found within the value in the application's **Modifier** field are converted to slashes and the value is truncated to 15 characters.

S0BE7<<change(Priority,"None","");
: When the application's **Priority** field contains the value None, this is converted to a null string.

S0C3E<<toIMDate(words(StartDate,1,1));
: This converts the first value (date) in the application's **StartDate** field from the format mm/dd/yyyy to the format mm/dd/yy.

S0C74<<toIMTime(words(StartDate,2));
: This converts the second value (time) in the application's **StartDate** field from the format hh:mm:ss am or hh:mm:ss pm to the format hh:mm, by dropping the seconds.

S0BEE<<translateWord(Status,"Approved","OPEN", "Pending","OPEN", "Working","OPEN", "Complete","CLOSED", "Rejected","CLOSED");

- If the value in the application's **Status** field is **Approved**, **Pending**, or **Working**, convert it to **OPEN**.
- If the value in the application's **Status** field is **Complete** or **Rejected**, convert it to **CLOSED**.

S0C09(8)<<translate(TroubleCode,".","/");
: All periods found within the value in the application's **TroubleCode** field are converted to slashes and the value is truncated to 8 characters.

S0B59(15)<<nullDefault(translate(Originator,", ","//"), "Helpdeskapp");
: All commas and spaces found within the value in your application's **Originator** field are converted to slashes and the value is truncated to 15 characters. If the Originator field is empty, a value of Helpdeskapp is mapped.

The following mappings are defined for propagate and transfer as specified in the transactions statement. No conversions are performed on the application's fields before being mapped.

```
Transactions=propagate,transfer;
 S0E01<<Description;
 00E02<<Detail;
 S0F52<<TicketNum;
```

Figure 6. Propagate and Transfer Transaction Mappings

The following mappings are defined for propagate as specified in the transactions statement.

```
Transactions=propagate;
 S0E0F<<"PROPAGATED RECORD FROM HELPDESKAPP";
 S1260<<"P?????";
 // ????? = GatewayID. Indicates this record was propagated
 // by this gateway.
 S14EF<<"?????";
 // ????? = GatewayID. Check-out indicator
```

Figure 7. Propagate Transaction Mappings

Where:

S0E0F<<**"PROPAGATED RECORD FROM HELPDESKAPP";**
: "PROPAGATED RECORD FROM HELPDESKAPP" is hardcoded into the Information/Management field.

S1260<<**"P?????";**
: Indicates this record was propagated by the gateway identified by the GatewayID value specified in substitution for P?????.

S14EF<<**"?????";**
: Is the checkout indicator; the record has been checked out by the gateway indicated in substitution for ?????. By being checked out, propagated records can only be updated by this gateway.

The following mappings are defined for the transfer operation as specified in the transactions statement. In this case no conversions are made before the mappings.

```
Transactions=transfer;
 S0E0F<<"TRANSFERRED RECORD FROM HELPDESKAPP";
 S1260<<"T?????";
```

Figure 8. Transfer Transaction Mappings

Where:

S0E0F<<"TRANSFERRED RECORD FROM HELPDESKAPP";
"TRANSFERRED RECORD FROM HELPDESKAPP" is hardcoded into the Information/Management field.

S1260<<"T?????";
Indicates that the record was transferred by the gateway identified by the GatewayID value specified in substitution for T?????.

Mapping Records from Information/Management to Your Application

The following mappings are defined for the Retrieve transaction as specified in the transactions statement. Information/Management data is mapped to your application's data.

```
Transactions=retrieve;
translateWord(change(nullDefault(S0B5A,"None"),"//",", ","/",","),
                   "NONE","None")>>Assignee;
S0E01>>Description(240);
translateWord(translate(nullDefault(S0B9C,"None"),"/"," "),
                   "ORGANIZATIO", "Organization",
                   "NONE","None")>>Organization;
nullDefault(stripLeading(fromIMPriority(S0BE7),"0"),"None")>>Priority;
translateWord(change(S0B59,"//",", ","/",", "),
                   "NONE","None")>>Originator;
translateWord(S0BEE,"INITIAL","Open",
                   "OPEN","Open", "CLOSED","Closed")>>Status;
translateWord(change(nullDefault(S0B5C,"None"),"//",", ","/",", "),
                   "NONE","None")>>Modifier;
translateWord(nullDefault(S0C09,"Unknown"),
                   "APPLICAT","Applications", "HARDWARE","Hardware",
                   "NETWORKS","Networks", "SOFTWARE","Software",
                   "UNKNOWN","Unknown") >> TroubleCode;
S0E02>>Detail(1000);
S0CA9>>Resource;
fromIMDate(S0C3E)" "translateWord(S0C74,"","","*",S0C74":00")>>StartDate;
S0CCF>>rnid;
```

Figure 9. Retrieve Transaction Mappings

Where:

translateWord(change(nullDefault(S0B5A,"None"), "//",",","/",","), "NONE","None")>>Assignee;
When the **S0B5A** field is empty, a value of "None" is mapped. If its value is "NONE", it is converted to "None", and all slashes are converted to commas, before being mapped into the application's **Assignee** field.

S0E01>>Description(240);
The first 240 characters of the Information/Management **S0E01** field are mapped into the application's **Description** field.

translateWord(translate(nullDefault(S0B9C,"None"),"/"," "), "ORGANIZATIO","Organization","NONE","None") >>Organization;
When the **S0B9C** field is empty, a value of "None" is mapped. "ORGANIZATIO" is converted to "Organization" and "NONE" is converted to "None". All slashes are converted to spaces before being mapped into the application's **Organization** field.

nullDefault(stripLeading(fromIMPriority(S0BE7),"0"),"None") >>Priority;
When the value in the **S0BE7** field is between 0 and 5, it remains unaltered. Otherwise a value of "5" is mapped. Leading zeros are removed and if the field is empty, the value "None" is mapped into the application's **Priority** field.

translateWord(change(S0B59,"//",", ","/",", "), "NONE","None")>>Originator;
If the value of the **S0B59** field is "NONE", it is converted to "None". All slashes are converted to commas before being mapped into the application's **Originator** field.

translateWord(S0BEE,"INITIAL","Open", "OPEN","Open", "CLOSED","Closed")>>Status;
The value "INITIAL" in the Information/Management **S0BEE** field is changed to "Open", the value "OPEN" to "Open", and the value "CLOSED" to "Closed" before being mapped into the application's **Status** field.

translateWord(change(nullDefault(S0B5C,"None"), "//",",","/", ","), "NONE","None") >>Modifier;

When the **S0B5C** field is empty, a value of "None" is mapped. If its value is "NONE", it is converted to "None". All slashes are converted to commas before being mapped into the application's **Modifier** field.

translateWord(nullDefault(S0C09,"Unknown"), "APPLICAT","Applications", "HARDWARE","Hardware", "NETWORKS","Networks", "SOFTWARE","Software", "UNKNOWN","Unknown") >>TroubleCode;

When there is a null value in the Information/Management **S0C09** field, it is replaced with the hardcoded value of "Unknown". All occurrences of "APPLICAT" in the field are replaced by "Applications", "HARDWARE" is replaced with "Hardware, "NETWORKS" with "Networks", "SOFTWARE" with "Software", and "UNKNOWN" with "Unknown".

S0E02>>Detail(1000);

The first 1000 characters of the Information/Management **S0E02** field are mapped into the application's **Detail** field

S0CA9>>Resource;

The value in the **S0CA9** field is mapped directly into the application's **Resource** field without being converted.

fromIMDate(S0C3E)" "translateWord(S0C74,"", "","*",S0C74":00")>>StartDate;

When the Information/Management time field (**S0C74**) is empty, it remains empty. Otherwise, seconds are added (:00) to the time already specified in hours and minutes (hh:mm). The Information/Management date field (**S0C3E**) is converted to mm/dd/yyyy format, and the time is appended to it before it is mapped into the application's **StartDate** field.

S0CCF>>rnid;

The value in the **S0CCF** field is mapped directly into the application's **rnid** field without being converted.

5

Completing Problem Service Configuration

This chapter explains how to complete the configuration of Problem Service.

Preparing the HLAPI Data Views on MVS

To define all the Information/Management record fields that Problem Service needs to access when performing the Problem Service operations, you need new HLAPI PIDTs and PIPTs or Information/Management data model records.

The HLAPI PIDTs that you need for the database records are create, update, and retrieve. Problem Service requires two fields to be defined in the HLAPI PIDTs or the data model's data view records:

- The create, update, and retrieve PIDTs or data view records must contain the Information/Management field that you specify in the GatewayIDField statement in the Problem Service configuration file. This field must be present in all Information/Management records processed by Problem Service.

  Note: The field specified in the GatewayIDField statement, in the Problem Service sample configuration file, is a new Information/Management record field. It has been defined in the provided PIDTs with Information/Management s-word index S1260 and p-word index P04E0.

- The create PIDT or data view record must also contain the Information/Management field specified in the IMCheckoutField statement in the Problem Service configuration. This is necessary for the propagate operation to always check out the propagated records.

How to proceed to prepare the new HLAPI PIDTs and PIPTs depends on whether the Information/Management records you are using with Problem Service are:

- Uncustomized (See “Preparing PIDTs and PIPTs for Uncustomized Records” for instructions on how to prepare the HLAPI tables.)
- Customized (See “Preparing PIDTs and PIPTs for Customized Records” on page 5-4 for information to assist you in preparing the HLAPI tables.)

To prepare the HLAPI tables on MVS you need the assistance of the Information/Management administrator.

Preparing PIDTs and PIPTs for Uncustomized Records

The Problem Service installation provides you with new PIDTs and PIPTs. These PIDTs and PIPTs have been customized and built to reflect the Information/Management field specifications used in the sample Problem Service configuration file and that match the uncustomized Information/Management record fields. They contain the fields of uncustomized Information/Management records as well as the two Information/Management fields required by Problem Service (GatewayIDField and IMCheckoutField).

The files containing these PIDTs and PIPTs are listed in Table 4 on page 5-3. These files are stored in the **/$INST_DIR/../include/$INTERP/InfoMgt/InfoGateway** directory unless you specified a different directory for header files during the installation of Problem Service.

Table 4. Information/Management PIDTs and PIPTs Shipped with Problem Service for Uncustomized Records

Filename	**Description**	**File Format**
BLMYGPC	Create PIDT	Binary
BLMYGPCP	Create PIPT	Binary
BLMYGPR	Retrieve PIDT	Binary
BLMYGPRP	Retrieve PIPT	Binary
BLMYGPU	Update PIDT	Binary
BLMYGPUP	Update PIPT	Binary

To use the provided HLAPI tables you need the assistance of the Information/Management administrator on MVS.

You must put these PIDTs and PIPTs in a report format table data set. This data set is a partitioned data set (PDS) on MVS with a record format of fixed block, a record length of 80, and a block size of 6160. For more information about defining a report format table data set, refer to the *Information/Management Planning and Installation Guide and Reference*.

To transfer the PIDTs and PIPTs to MVS, you can use, for example, the File Transfer Protocol (**ftp** command) method if your workstation has a TCP/IP link to the MVS host. After establishing an ftp connection to the MVS host, issue the subcommand `type binary` to ensure that the tables are transferred in binary format. Use the `put` subcommand to store the tables in the PDS.

The PIDT and PIPT data sets must be allocated to the DDNAME that is associated with the Information/Management report format tables or referenced in the Information/Management session member used to initialize the API session.

For more information on how Information/Management uses PIDTs and PIPTs, refer to the *Information/Management Application Program Interface Guide*.

Preparing PIDTs and PIPTs for Customized Records

Use the MVS system where Information/Management is running to create new PIDTs and PIPTs. Customizing these tables requires experience with the Information/Management product and knowledge of how records are customized in Information/Management. If you do not have experience with this product, you need the direct assistance of the Information/Management administrator to customize these tables on MVS. The Information/Management administrator will also have the necessary information about the characteristics of the fields of the customized records in Information/Management.

Customizing HLAPI Database Tables

This section provides you with a general overview on how:

- To customize the HLAPI tables
- To configure Information/Management
- To use data model records instead of PIDTs and PIPTs

Information/Management provides you with an MVS utility, called BLGUT8, that you use to build each customized PIDT and corresponding PIPT in Information/Management. The BLGUT8 utility requires the following input:

- Input statements that specify the record fields for the PIDT.
- Information/Management assisted entry panels (in offloaded format using utility BLGUT6F) for the fields specified in the PIDT.
- Information/Management dictionary entries defining the characteristics of the fields specified in the PIDT.

  Note: For more information on assisted entry panels and the Information/Management dictionary, refer to the *Information/Management Panel Modification Facility Guide and Reference*

The BLGUT8 utility generates PIDTs and PIPTs that HLAPI uses for your application. For detailed descriptions on customizing HLAPI tables, configuring Information/Management, using data model records, and the BLGUT8 utility, refer to the *Information/Management Application Program Interface Guide*.

The Problem Service installation provides you with sample files, which you can use as examples when you are preparing the input required by the BLGUT8 utility to build your customized HLAPI tables. These files are stored in the **/$INST_DIR/../include/$INTERP/InfoMgt/InfoGateway** directory unless you specified a different directory for header files during the installation of Problem Service.

This section describes how you can use the provided sample files during the customization process. The sample files are listed in Table 5.

Table 5 (Page 1 of 2). Information/Management Sample Files To Use When Building Customized PIDTs and PIPTs

Filename	**Description**	**File Format**
BLMYGPCS	Sample file providing an example of the input statements for the create PIDT for uncustomized database records. These statements include specifications for the Problem Service required fields (the Information/Management fields specified in the GatewayIDField and IMCheckoutField statements in the sample Problem Service configuration file).	ASCII
BLMYGPRS	Sample file providing an example of the input statements for the retrieve PIDT for uncustomized database records. These statements include the specification for the Problem Service required field (the Information/Management field specified in the GatewayIDField statement in the sample Problem Service configuration file).	ASCII

Table 5 (Page 2 of 2). Information/Management Sample Files To Use When Building Customized PIDTs and PIPTs

Filename	**Description**	**File Format**
BLMYGPUS	Sample file providing an example of the input statements for the update PIDT for uncustomized Information/Management records. The statements include the specification for the Problem Service required field (the Information/Management field specified in the GatewayIDField statement in the sample Problem Service configuration file).	ASCII
BLMYGGID	Sample assisted entry panel in offloaded format. It defines the characteristics of the field specified in the GatewayIDField statement in the Problem Service configuration file.	Binary
BLMYDICT	Partial Information/Management dictionary that defines a dictionary entry for the GatewayIDField using s-word index S1260 and p-word index P04E0.	Binary

The following guidelines describe how you can use the sample files when you are preparing the input for the BLGUT8 utility:

- When you are preparing the PIDT input statements for the various database record PIDTs, include the input statements to specify the two fields required by Problem Service. Use the sample files **BLMYGPCS**, **BLMYGPUS**, and **BLMYGPRS** to see how to specify the information for these fields. Table 5 on page 5-5 provides a description of the contents of these files.
- When you are preparing the assisted entry panels for the fields in the PIDTs, you also need to provide an assisted entry panel for the field specified in the GatewayIDField statement in the Problem Service configuration file.

  If you are using this field as specified in the sample configuration file, you can use the sample assisted entry panel provided in file **BLMYGGID**. If you specified a different field, you need to define an assisted entry panel for that field.

If you want to use the sample assisted entry panel:

- Transfer the **BLMYGGID** file to the Information/Management system on MVS and place it in the Information/Management panel PDS that will be used by the BLGUT8 utility.
- Ensure that the input statements of the PIDTs you are creating associate this panel to the GatewayIDField statement.

▪ When you are preparing the Information/Management dictionary, remember to include a dictionary entry for the field specified in the GatewayIDField statement in the Problem Service configuration file.

When you use this field as specified in the sample configuration file, you can use the partial Information/Management dictionary provided in the **BLMYDICT** file. The Information/Management product provides a utility called BLGUT5 that enables you to load a partial dictionary into the Information/Management dictionary.

To use the **BLMYDICT** file:

1. Transfer the **BLMYDICT** file to the Information/Management system on MVS and place it in a data set that can be used by the BLGUT5 utility.
2. Use the BLGUT5 utility to load this partial dictionary into the Information/Management dictionary that will be used by the BLGUT8 utility. For information on how to use the BLGUT5 utility refer to the *Information/Management Operation and Maintenance Reference*

Updating the Services File

After you complete the customization of the Problem Service configuration file, update the services file on your workstation to reflect the settings you specified in the configuration file.

AIX Workstation /etc/services File

In the **/etc/services** file, define the entry for the service name you specified in the InfoGatewayService statement and associate an available TCP/IP port number for this service name.

For example, if you customized the statement with the following service name:

```
InfoGatewayService="infogateway";
```

The entry in the **/etc/services** file would be as follows:

```
infogateway 1453/tcp
```

Select a TCP/IP port number (for example, *1453*) for the service name that is unique and available on your system.

Windows NT Workstation Services File

In your IP services file, define the entry for the service name you specified in the InfoGatewayService statement and associate an available TCP/IP port number for this service name.

For example, if you customized the statement with the following service name:

```
InfoGatewayService="infogateway";
```

The entry in the services file would be as follows:

```
infogateway 1453/tcp
```

Select a TCP/IP port number (for example, *1453*) for the service name that is unique and available on your system.

6

Running Problem Service

Problem Service can be started after installation and configuration is complete, and HLAPI has been invoked successfully.

Starting Problem Service

For AIX, Problem Service is started by running the **gw_nxd** executable file. Other daemon sessions are automatically started by the base process according to what was specified for the session statements in the Problem Service configuration file.

You can start Windows NT system services by opening the control panel folder and double-clicking on the services icon. The name will be displayed in the services list as `TME 10 INFO Gateway`. Click on this name, then click on the start button.

You must start Problem Service from an administrator user ID in the Tivoli Management Region (TMR) where Problem Service is installed to use reverse assignment and monitor operations.

Stopping Problem Service

For AIX, use the **shutdown** Problem Service function or locate the process ID of the main Problem Service process and use the **kill** command to stop the daemon. The main process will stop the subprocesses. For more information on the shutdown function, refer to “Shutdown” on page A-8.

You can stop Windows NT system services by opening the control panel folder and double-clicking on the services icon. The name will be displayed in the services list as `TME 10 INFO Gateway`. Click on this name, then click on the stop button.

Logging with Problem Service

Problem Service has the ability to write error, trace, and informational messages to a log file. There are several characteristics of the logging function that can be customized. A list of these, along with their default values, are:

Log name This is the name of the log file. The default log file name used is infogw.log and is created in the current directory.

Log size This is the maximum size of the log file. The default log size is set to 250000 bytes. Upon reaching this size the current active log is archived and a new log file is started. One archived backup log is maintained and uses the name of the log file with .bak concatenated to the end. If the default log name is used, the backup file would have the name infogw.log.bak.

Archiving takes place by removing the existing backup file and then renaming the current active log file to the backup name. The new log file is then created.

Log state This is the state of the logger (ON or OFF). The logger is ON by default.

Log level This is the level of logging to be performed. The logger has three levels of logging:

Level 1 Only messages listed as errors are written to the log file

Level 2 Error messages plus trace messages are written to the log file

Level 3	Error, trace, and informational messages are written to the log file

The log level is set to level 3 by default, which causes all messages to be logged.

The logging characteristics can be customized by setting and exporting the following environment variables:

GWLOGNAME	Specifies the name of the log file.
GWLOGSIZE	Specifies the maximum size of the log file in bytes. The smallest value that can be specified is 5000 bytes. A smaller value results in 5000 bytes being used as the maximum size.
GWLOGSTATE	Specifies the state of the logger. The logger is disabled if the value of this variable is set to OFF. Other values cause the logger to be ON.
GWLOGLEVEL	Sets the level of logging. The valid values are 1, 2, or 3. A value less than 1 causes 1 to be used as the level and a value more than 3 causes 3 to be used as the level.

The environment variables must be exported from a parent session of the one used by Problem Service.

Each output to the log file will contain the following information:

```
Date     Time     PID   MessageID     MessageText
```

An example output to the log is:

```
01/10/97 16:32:35 18454 APAGM034I Process configuration file blymgc.cfg.
01/10/97 16:32:38 3610  APAGT003I Gateway Reverse Assignment process started
01/10/97 16:32:38 3610  APAGT028I Connect to Information/Management
```

This log file, along with the HLAPI client log and the HLAPI client probe log, can be useful in identifying and locating problems. Information about the two HLAPI client log files can be found in the *Information/Management Client Installation and User's Guide*.

A

Appendix A. Application Programming Information

This appendix is for the programmer who is developing an application that will invoke the Problem Service operations. Refer to the TME 10 Application Development Environment (TME 10 ADE) publications for more information about developing an application in the TME environment.

Copying the Samples and Files

Copy the samples and files you need to compile calls to the APIs.

- Create a test directory (example: **/home/userid/tivtest**).
- Change directory to the new test directory.
- Copy the files into the new test directory. The files are in **$INST_DIR/../include/$INTERP/InfoMgt/InfoGateway** unless you specified a different directory for header files during the installation of Problem Service.

Compiling and Link Editing Your Code

Compile and link edit your code with the Problem Service APIs.

- Change to your test directory.
- Edit the Makefile for your setup:
  - Change **TOP** (directory where TME 10 is installed)

  - Change **HERE** (your test directory)
  - For Windows NT only, change **TOOLROOT** (your compiler directory)
  - Change references to tester, tester.c, and tester.o to the appropriate names for your application if you have written your own program that uses the Problem Service operations.
  - Change the compiler commands and libraries to those for your compiler. For AIX the sample Makefile uses the xlC compiler and the IBMcset libraries. For Windows NT, the sample tester.mak uses the IBM VisualAge for C++ for Windows compiler.

- For AIX workstations, enter `ln -s /home/userid/tivtest tivoli` to set up a symbolic link used by the make program to find the files you copied.
- Compile and link edit your program. The sample C program, tester.c, uses the gnu make (gmake) program. For AIX workstations, enter `gmake test` to compile and link the tester.c program using the gnu make program. For Windows NT workstations, enter `make -f tester.mak tester.exe` to compile and link the tester.c program using the gnu make program.

Refer to the TME 10 Application Development Environment publications for more information about creating an application in the TME 10 environment.

Interface Definition Language Data Types

The following are the interface definition language (IDL) data types used by the Problem Service API:

string	Null terminated character string
unsigned long	32 bit integer
sequence	A one dimensional array of elements. The Common Object Request Broker Architecture (CORBA) specification defines the sequence data type for operations that accept or return a set of data

structures. TME 10 provides a library of functions for manipulating sequences.

Following is an example of a sequence that can be used in coding Problem Service operations:

```
struct GWAttr {
  string name;    // name of data field
  string value;   // value of data field
};

typedef sequence <GWAttr>GWAttrList;
```

The GWAttrList sequence contains a list of elements that represent a record. Each element has a name and value. The name identifies the data field name and the value contains the value associated with the data field.

Coding Examples for Problem Service Operations

The following code fragments demonstrate how to code the Problem Service operations in an application program. For more detail and to see each example in context, refer to the Bash shell script examples in the **sample** file and the C code examples in the **tester.c** file that are shipped with Problem Service.

Checkin

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
idlcall $OID InfoGW::checkin \"00000311\"
```

C example:

```
/****************************************************/
 /* Variables                                        */
 /****************************************************/
 Environment     ev;
 Object          oid = OBJECT_NIL;

 /****************************************************/
 /* Get the object id                                */
 /****************************************************/
 oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
 /****************************************************/
 /* Perform the transaction                          */
 /****************************************************/
 t_InfoGW_checkin(oid, &ev, Trans_none, "00000311");
```

Checkout

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
idlcall $OID InfoGW::checkout \"00000311\"
```

C example:

```
/****************************************************/
 /* Variables                                        */
 /****************************************************/
 Environment     ev;
 Object          oid = OBJECT_NIL;

 /****************************************************/
 /* Get the object id                                */
 /****************************************************/
 oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
 /****************************************************/
 /* Perform the transaction                          */
 /****************************************************/
 t_InfoGW_checkout(oid, &ev, Trans_none, "00000311");
```

Delete

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
idlcall $OID InfoGW::delete \"00000311\"
```

C example:

```
/*****************************************************/
 /* Variables                                         */
 /*****************************************************/
 Environment     ev;
 Object          oid = OBJECT_NIL;

 /*****************************************************/
 /* Get the object id                                 */
 /*****************************************************/
 oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
 /*****************************************************/
 /* Perform transaction                               */
 /*****************************************************/
 t_InfoGW_delete(oid, &ev, Trans_none, "00000311");
```

Ping

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
idlcall $OID InfoGW::ping
```

C example:

```
/*****************************************************/
/* Variables                                         */
/*****************************************************/
Environment     ev;
Object          oid = OBJECT_NIL;

/*****************************************************/
/* Get the object id                                 */
/*****************************************************/
oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
/*****************************************************/
/* Perform transaction                               */
/*****************************************************/
t_InfoGW_ping(oid, &ev, Trans_none);
```

Propagate

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
RNID1=`idlcall $OID InfoGW::propagate { 2 \
     {\"Originator\"\"Smith,Bill\"}\
     {\"Status\"\"Pending\"}} \"Helpdeskapp\"`
```

C example:

```
  /****************************************************/
  /* Variables                                        */
  /****************************************************/
  Environment  ev;
  Object       oid = OBJECT_NIL;
  GWAttrList   gwattrlist;
  GWAttr       tmpattr;
  char         * rnid;

  /****************************************************/
  /* Get the object id                                */
  /****************************************************/
  oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
  /****************************************************/
  /* Initialize the sequence of name value pairs      */
  /****************************************************/
  seq_init((sequence_t * ) &gwattrlist );
  /*********************************************************/
  /* Set the element values and append them to the sequence */
  /*********************************************************/
  tmpattr.name  = ml_ex_strdup("Originator");
  tmpattr.value = ml_ex_strdup("Smith,Bill");
  seq_add( (sequence_t *)&gwattrlist,&tmpattr,sizeof(GWAttr) );
  tmpattr.name  = ml_ex_strdup("Status");
  &tmpattr.value = ml_ex_strdup("Pending");
  seq_add( (sequence_t *)&gwattrlist,&tmpattr,sizeof(GWAttr) );
  /**********************************************************/
  /* Perform the transaction and save the record id         */
  /**********************************************************/
  rnid=t_InfoGW_propagate(oid,&ev,Trans_none,&gwattrlist,"Helpdeskapp");
  /**********************************************************/
  /* Free the input sequence's buffer                       */
  /**********************************************************/
  seq_free_buffer((sequence_t *) &gwattrlist);
```

Retrieve

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
RESULT=`idlcall $OID InfoGW::retrieve \"00000311\"\"Helpdeskapp\"`
```

C example:

```
  /*****************************************************/
  /* Variables                                         */
  /*****************************************************/
  Environment    ev;
  Object         oid = OBJECT_NIL;
  GWAttrList     myrec;

  /*****************************************************/
  /* Get the object id                                 */
  /*****************************************************/
  oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
  /*****************************************************************/
  /* Perform the transaction                                       */
  /*****************************************************************/
  myrec=t_InfoGW_retrieve(oid,&ev,Trans_none,"00000311","Helpdeskapp");
```

Search

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
RESULT2=`idlcall $OID InfoGW::search { 2 \
     {\"Originator\"\"Smith,Bill\"}
     {\"Status\"\"Pending\"}} \"Helpdeskapp\"`
```

C example:

```
/****************************************************/
/* Variables                                        */
/****************************************************/
Environment       ev;
Object            oid = OBJECT_NIL;
GWAttrList        gwattrlist;
GWAttr            tmpattr;
SearchResultList  mysearchlist;

/****************************************************/
/* Get the object id                                */
/****************************************************/
oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
/*************************************************************/
/* Initialize the sequence of name value pairs              */
/*************************************************************/
seq_init((sequence_t * ) &gwattrlist);
/*************************************************************/
/* Set the element values and append them to the sequence    */
/*************************************************************/
tmpattr.name  = ml_ex_strdup("Originator");
tmpattr.value = ml_ex_strdup("Smith,Bill");
seq_add( (sequence_t *)&gwattrlist, &tmpattr, sizeof(GWAttr) );
tmpattr.name  = ml_ex_strdup("Status");
tmpattr.value = ml_ex_strdup("Pending");
seq_add( (sequence_t *)&gwattrlist, &tmpattr, sizeof(GWAttr) );
/*************************************************************/
/* Perform the transaction                                   */
/*************************************************************/
mysearchlist = t_InfoGW_search(oid, &ev, Trans_none,
                               &gwattrlist, "Helpdeskapp");
/*************************************************************/
/* Free the input sequence's buffer                          */
/*************************************************************/
seq_free_buffer((sequence_t *) &gwattrlist);
```

Shutdown

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
idlcall $OID InfoGW::shutdown
```

C example:

```
/****************************************************/
/* Variables                                        */
/****************************************************/
Environment      ev;
Object           oid = OBJECT_NIL;

/****************************************************/
/* Get the object id                                */
/****************************************************/
oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
/****************************************************/
/* Perform transaction                              */
/****************************************************/
t_InfoGW_shutdown(oid, &ev, Trans_none);
```

Transfer

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
RNID=`idlcall $OID InfoGW::transfer { 2 \
    {\"Originator\"\"Smith,Bill\"} \
    {\"Status\"\"Pending\"}} \Helpdeskapp\"`
```

C example:

```
/****************************************************/
/* Variables                                        */
/****************************************************/
Environment  ev;
Object       oid = OBJECT_NIL;
GWAttrList   gwattrlist;
GWAttr       tmpattr;
char         * rnid;

/****************************************************/
/* Get the object id                                */
/****************************************************/
oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
/*****************************************************************/
/* Initialize the sequence of name value pairs                   */
/*****************************************************************/
seq_init((sequence_t *)  &gwattrlist);
/*****************************************************************/
/* Set the element values and append them to the sequence.       */
/*****************************************************************/
tmpattr.name  = ml_ex_strdup("Originator");
tmpattr.value = ml_ex_strdup("Smith,Bill");
seq_add( (sequence_t *) &gwattrlist, &tmpattr, sizeof(GWAttr) );
tmpattr.name  = ml_ex_strdup("Status");
tmpattr.value = ml_ex_strdup("Pending");
seq_add( (sequence_t *) &gwattrlist, &tmpattr, sizeof(GWAttr) );
/*****************************************************************/
/* Perform the transaction and save the record id                */
/*****************************************************************/
rnid=t_InfoGW_transfer(oid,&ev,Trans_none, &gwattrlist,"Helpdeskapp");
/*****************************************************************/
/* Free the input sequence's buffer                              */
/*****************************************************************/
seq_free_buffer((sequence_t *)  &gwattrlist);
```

Update

Script example:

```
OID=`wlookup -r InfoMgtGW Info_GW`
idlcall $OID InfoGW::update { 2 \
     {\"Status\"\"Closed\"}
     {\"rnid\"\"311\"}} \"Helpdeskapp\"
```

C example:

```
/****************************************************/
/* Variables                                        */
/****************************************************/
Environment  ev;
Object       oid = OBJECT_NIL;
GWAttrList   gwattrlist;
GWAttr       tmpattr;

/****************************************************/
/* Get the object id                                */
/****************************************************/
oid = dir_lookup_instance("InfoMgtGW", "Info_GW");
/***************************************************************/
/* Initialize the sequence of name value pairs                */
/***************************************************************/
seq_init((sequence_t * )  &gwattrlist);
/***************************************************************/
/* Set the element values and append them to the sequence.     */
/***************************************************************/
tmpattr.name  = ml_ex_strdup("Status");
tmpattr.value = ml_ex_strdup("Closed");
seq_add( (sequence_t *) &gwattrlist,&tmpattr,sizeof(GWAttr) );
tmpattr.name  = ml_ex_strdup("rnid");
tmpattr.value = ml_ex_strdup("00000311");
seq_add( (sequence_t *) &gwattrlist,&tmpattr,sizeof(GWAttr) );
/***************************************************************/
/* Perform the transaction                                     */
/***************************************************************/
t_InfoGW_update(oid, &ev, Trans_none,  &gwattrlist, "Helpdeskapp");
/***************************************************************/
/* Free the input sequence's buffer                            */
/***************************************************************/
seq_free_buffer((sequence_t *) &gwattrlist);
```

TME 10 Application Development Environment Exceptions

TME 10 ADE exceptions are used to return error information to your application. Problem Service defines the ExInfoGateway exception. Refer to the TME 10 Application Development Environment documentation for more information on these exceptions.

ExInfoGateway Exception

The following data is inherited from ExException:

string type_name
Name of exception type

string catalog
Name of message catalog

long key Message catalog key

string default_message
Default message, when the message catalog is unavailable

long stamp
Date stamp

Msgcontext Msg_context
Context of exception raised

The following are ExInfoGateway data:

string message
Name of C++ exception

HICAReturnCode
Information/Management API return code

HICAReasonCode
Information/Management API reason code

Examples of Gateway Exceptions

```
Try {
  t_InfoGW_checkout(oid, &ev, Trans_none,recordID);
}
Catch(ExInfoGateway,ex){//catches Gateway exceptions and
                          //any exceptions derived from
                          //ExInfoGateway
}
Catch(ExException,ex){  //catches ExExceptions and any
                          //exceptions derived from ExException
}
CatchAll() {            //catches all exceptions
}
```

B

Appendix B. Customizing User Exit Routines for the Daemon

Problem Service assumes that all data is character data. Some data conversions, such as the truncation of data, are supported. Problem Service does not know the Information/Management database record structure, and the Information/Management HLAPI does not automatically convert data, so you must define how truncations and other conversions are to be performed.

Some types of mapping syntax enable you to define fields and rules for mappings for each type of transaction. The mapping rule for a particular field can be different for different transactions.

Supported Data Conversions

Here is a list of the mapping conversions that are supported, followed by a description of each listed item:

- Truncation
- Convert one character to another character
- Convert specific field value to another value
- Date/time conversion
- Freeform text
- Default data
- Field combining (concatenation)

- Substring and sub-word
- Exit routines

Truncation

Data can be truncated to a specified length. For example, assignee in an application's record is 90 characters, while assignee in an Information/Management database record is only 15 characters, so you can choose to truncate after the first 15 characters.

Convert One Character to Another Character

An application's field could contain blanks and commas while the corresponding Information/Management field might allow only one word and not allow commas. The blanks and commas in the field are converted to a specified character that you can choose.

The Information/Management record field may allow multiple words (for example, first and last name). In this case, the blanks and commas would be converted to a specified separator character. This allows the HLAPI client to indicate that the data contains multiple words (separator character separates each word).

Convert Specific Field Value to Another Value

An application's field might allow hardcoded values that are different from the corresponding Information/Management problem fields. Each value that does not match is converted to a specified value.

Date/Time Conversion

An application's time stamp can be a combination of date and time. The date part and the time part are put into the Information/Management record's respective date field and time field. Information/Management enables you to choose which external date and time formats to use (exit routine is used to convert from internal to external and vice versa).

The HLAPI client accepts dates and times in external format (this could be different for each Information/Management site). Problem Service allows the specification of a C exit routine to convert the application's record date/time into dates and times to be given to Information/Management.

Note: This means that a C compiler is a prerequisite if you want to use your own user exits.

The date and time conversion exit can manipulate the date and time to support time zone differences between Information/Management and your applications.

Freeform Text

An application's freeform textual field is converted to Information/Management freeform text. An application's text data might be just a stream of characters with no indications of new lines. In this case, specify the length of the corresponding Information/Management text line so that your application's text can be split into Information/Management text lines. When returning freeform text, the text lines can be converted to a data stream before being given to your application.

Default Data

You can specify hardcoded data for a field, either on an unconditional basis or only if the source data field is empty.

Field Combining (Concatenation)

You can combine multiple fields into one target field. For example, several of your application's fields might map into just one Information/Management field.

Substring and Sub-Word

You can choose to map only a part of a field into the target field.

Exit Routines

An exit routine can be specified to perform whatever conversions you choose. This exit routine must be written in the C programming language.

Specifying User Exits for Conversions

You can specify exit routines to perform data conversions. These can be routines you write yourself or those provided by Problem Service.

If you want to write your own exit routine, code it as shown in the following example:

```
char* main(int argumentCount, const char ** argumentArrayPointer);
```

To ensure that control returns to the mapping facility after the invocation of a user exit, it must be linked with the entry point as `main` and not a compiler-generated routine. Refer to the link options of the compiler you are using for instructions.

Copy the newly created user exit routine to the directory where the gw_nxd daemon executable exists, so that Problem Service can use it.

Several user exits to perform data conversions are provided. The first argument for all of the exits is *data*, and is either absent, a field name, a literal string, or another subroutine specification. This is also true for any other argument requiring a string.

Table 6 on page B-5 is a list of the supplied user exits. Each user exit with examples of use is described following the table.

Table 6. Supplied User Exits

User Exit	Description
change	Returns specified string changes.
fromIMDate	Converts a date format year value.
fromIMPriority	Maps priority values.
fromIMTime	Converts a military time format.
nullDefault	Returns a specified value when the target is null.
stripLeading	Strips leading characters.
subString	Returns a specified substring.
toIMDate	Converts a date format year value.
toIMTime	Converts a time format to military time format.
translate	Returns a specified character translation.
translateWord	Returns a specified word translation.
words	Returns a specified substring.

change

This user exit returns specified string changes of the target *data*. The format is:

```
change(data, sourceWord1, targetWord1,
             sourceWord2, targetWord2, ...)
```

The string given by *sourceWord1,* where found within *data*, is changed to the string given by *targetWord1*. It then changes the string given by *sourceWord2,* where found in the result of the first operation, to the string given by *targetWord2*. This continues until all source strings have been processed. If the last matching target string is missing, it defaults to null.

For example, if *fieldName* is *abcdefghijklm* then:

```
change(fieldName,"abc","def","def","ghi") returns "ghighighijklm"
change("abcdefghijklm","def")            returns "abcghijklm"
```

fromIMDate

This user exit converts a *yy* date format to a *yyyy* date format. The format is:

```
fromIMDate(IMDate)
```

A date in the format `mm/dd/yy` is converted to `mm/dd/yyyy`, where year characters 50 through 99 represent the years 1950 through 1999 and year characters 00 through 49 represent the years 2000 through 2049.

For example,

```
fromIMDate("12/05/49") returns "12/05/2049"
```

fromIMPriority

This user exit maps priority values. The format is:

```
fromIMPriority(IMPriority)
```

It maps values 6 through 99 to 5, while not altering values 0 through 5.

For example,

```
fromIMPriority("21") returns "5"
fromIMPriority("0")  returns "0"
```

fromIMTime

This user exit converts a military time format. The format is:

```
fromIMTime(IMTime)
```

The military time format of `hh:mm` is converted to a time in the format `hh:mm:ss am` or `hh:mm:ss pm`, by adding a seconds field of 00 and am or pm.

For example,

```
fromIMTime("13:34") returns "01:34:00 pm"
```

nullDefault

This user exit returns a specified value when the target is null. The format is:

```
nullDefault(data, defaultValue)
```

The value *defaultValue* is returned when *data* is null. Otherwise, the value for *data* is returned.

For example, if *fieldName* is *abc* then:

```
nullDefault(fieldName,"default") returns "abc"
nullDefault(,fieldName)          returns "abc"
nullDefault(,)                   returns ""
```

stripLeading

This user exit strips the leading characters from the target *data*. The format is:

```
stripLeading(data, stripCharacters)
```

For example, if *fieldName* is *0000200* then:

```
stripLeading(fieldName,"0")       returns "200"
stripLeading(fieldName)           returns "0000200"
stripLeading("000000","0")        returns ""
stripLeading("wordwordzz","word") returns "zz"
```

subString

This user exit returns a specified substring of the target *data*. The formats are:

```
subString(data, startPosition)
subString(data, startPosition, length)
subString(data, startPosition, length, padCharacter)
```

Where:

startPosition	Starting index position of the substring. If the index is beyond the end of the string, the function returns a null string.
length	The length of the substring. If the substring extends beyond the end of the string, the substring is padded with the character given by the *padCharacter*

argument. If length is not specified, the substring goes from the starting position to the end of the string.

padCharacter The character to use as padding if the substring extends beyond the end of the string. The default pad character is a single space.

For example, if *fieldName* is *abcdef* then:

```
substr(fieldName,2,3)           returns "bcd"
substr(fieldName,4,5)           returns "def  "
substr(substr("abcdef",2,3),2,1) returns "c"
substr("abcdef",7,1)            returns " "
```

toIMDate

This user exit converts a *yyyy* date format to a *yy* date format. The format is:

```
toIMDate(foreignDate)
```

A date in the format `mm/dd/yyyy` is converted to `mm/dd/yy`, where year characters 50 through 99 represent the years 1950 through 1999 and year characters 00 through 49 represent the years 2000 through 2049.

For example,

```
toIMDate("01/31/1950") returns "01/31/50"
```

toIMTime

This user exit converts a time format to military time format. The format is:

```
toIMTime(foreignTime)
```

A time in the format `hh:mm:ss am` or `hh:mm:ss pm` is changed to military time `hh:mm`, by dropping the seconds.

For example,

```
toIMTime("12:34:56 am") returns "00:34"
```

translate

This user exit returns a specified character translation of the target *data*. The format is:

```
translate(data, inputCharacters, outputCharacters)
```

The characters given by *inputCharacters*, where found within *data*, are changed to the characters given by *outputCharacters*. If not specified, *outputCharacters* defaults to spaces.

For example, if *fieldName* is *a,b,c,d* then:

```
translate(fieldName,","," ")    returns "a b c d"
translate("a,b,c,d",fieldName)  returns "       "
```

translateWord

This user exit returns a specified word translation of the target *data*. The format is:

```
translateWord(data, sourceWord1, targetWord1,
                    sourceWord2, targetWord2, ...)
```

The string given by *sourceWordN* is changed to the string given by *targetWordN*, if found within *data*. If the last matching target is missing, it defaults to null. If *sourceWord1* is *, any result for *data* is a match.

For example, if *fieldName* is *word* then:

```
translateWord(fieldName,fieldName,"bird") returns "bird"
translateWord("word","word")              returns ""
```

words

This user exit returns a specified substring of the target *data*. The format is:

```
words(data, firstWord)
words(data, firstWord, numberOfWords)
```

It begins with the word in the word index position given by *firstWord* (words are separated by spaces). If the index given by *firstWord* is not valid, the function returns a null string. The *numberOfWords* argument can be used to specify how many words to include in the substring. If the *numberOfWords* argument is not specified, all the words to the end of the string are included in the substring. The substring contains all the word separators (spaces) that are included in the original string.

For example, if *fieldName* is *a b c d* then:

```
words(fieldName,2,1)               returns "b"
words(subString(fieldName,3,3),2,2) returns "c"
words("a b c d",5,1)               returns ""
```

Index

GC31-8543-02