High -Level Screen Management Intrinsics Library Reference Manual

HP 3000 MPE/iX Computer Systems

Edition 1

Manufacturing Part Number: 32424-90002
E1187

U.S.A. November 1987

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for direct, indirect, special, incidental or consequential damages in connection with the furnishing or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c) (1,2).

Acknowledgments

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Preface

High-Level Screen Management Intrinsic Library (Hi-Li) Reference Manual

Printed in USA
HP Part No. 32424-90002
Edition E1187
Printed Nov 1987

First Edition Nov 1987

32424A.00.08

PREFACE

This publication is the reference manual for HP's High-Level Screen Management Intrinsic Library (Hi-Li). Hi-Li comprises a group of twelve intrinsics that facilitate the transfer of data between an application and most HP terminals and data capture devices on the HP 3000 computer system. Hi-Li provides the programmatic interface to terminals and forms.

Hi-Li is intended to be used by programmers in any of the supported languages, which include COBOL, FORTRAN, and Pascal. The manual is organized to start with a discussion of how Hi-Li works, followed by detailed information about each intrinsic and a description of each data transfer method. The appendices contain error messages, sample programs,
and information on supported terminals.

In order to use Hi-Li effectively, you should be familiar with FORMSPEC, the VPLUS/V forms file designer. Hi-Li uses forms created in FORMSPEC. You may also need the manual for the programming language in which you are coding. The manuals in the following list contain all the information you might need as a supplement to this manual:

* *HP Data Entry and Forms Management System (VPLUS/V) Reference Manual*

* *MPE Commands Reference Manual*

* *MPE Intrinsics Reference Manual*

* *MPE Software Pocket Guide*

* *Using Files*

* *Native Language Support Reference Manual*

You may also refer to the reference manual for the programming language you are using, and the reference manual for your terminal. s

Chapter 1 Introduction

This chapter introduces the High-Level Screen Management Intrinsic Library (Hi-Li) and discusses how the intrinsics work with Hewlett-Packard's screen management system. The topics include:

* What the Hi-Li intrinsics are
* How they work with HP's screen management system to transfer data between the screen and the application
* The features of Hi-Li
* Hi-Li's requirements
* How this manual is organized

The High-Level Screen Management Intrinsic Library

The High-Level Screen Management Intrinsic Library (Hi-Li) comprises a number of callable procedures that link your application to HP's screen management facility. The screen management facility controls the transfer of information between an application and the screen at run time
by providing an interface with the system-level input and output (I/O) operations. Figure 1-1 illustrates the relationship between Hi-Li, the application, and the terminal with its I/O operations.

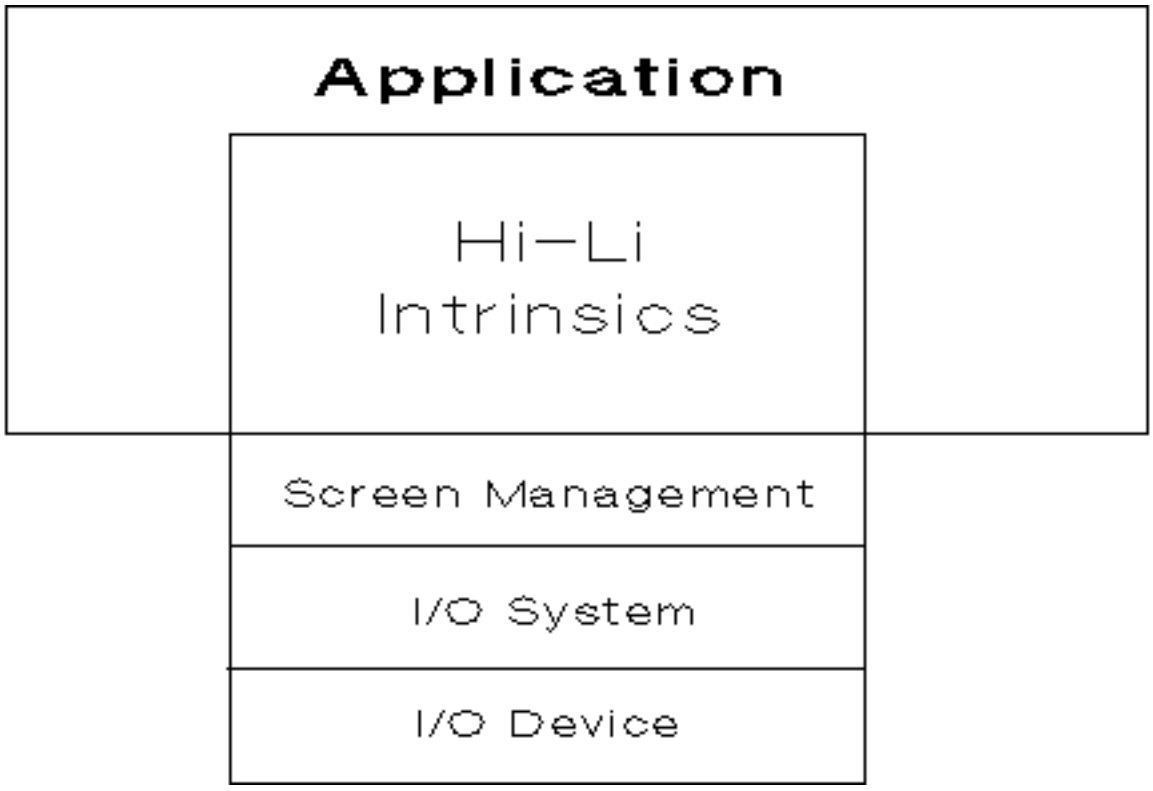

Figure 1-1. Hi-Li and the Screen Management Facility

Each intrinsic performs several system-level functions. This allows you to keep your application design simple and reduces code maintenance resulting from system-level changes. As the screen management facility handles system-level details, you can concentrate on the actual functions
your application needs to perform instead of the requirements for implementing those functions.

Part of the screen management facility is the FORMSPEC/V utility, with which you can design interactive user interfaces called "forms". Hi-Li intrinsics act as an interface between the application, the terminal and the forms, controlling the transfer of data to and from the terminal. You can use Hi-Li intrinsics to manage the display of forms on a terminal screen and the entry, collection, and validation of data via these forms.
Figure 1-2 shows the flow of data from point to point.

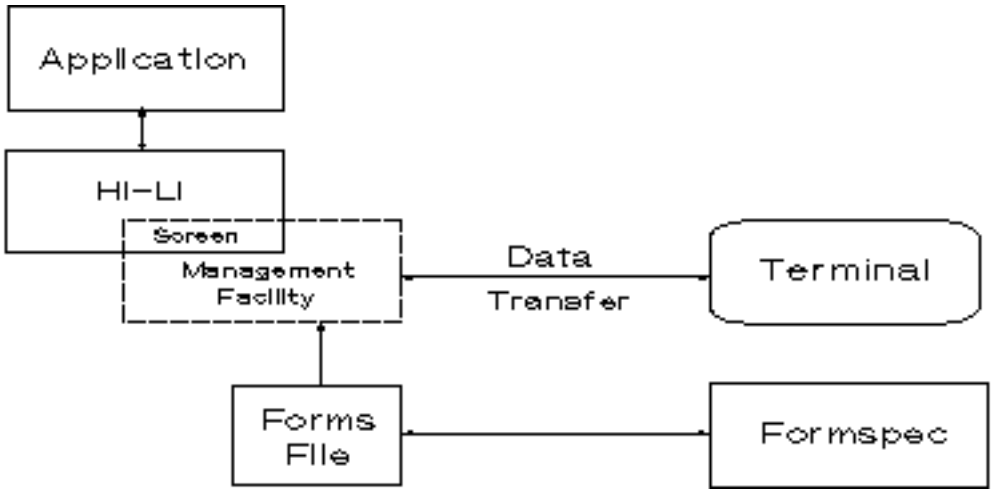

Figure 1-2. Hi-Li and the Transfer of Data

Fetures of HI-LI

The High-Level Screen Management Intrinsic Library (Hi-Li) has the following features:

* **High-level calls:** Hi-Li intrinsics combine the most commonly used low-level terminal input/output procedures into 12 high-level calls. Application development is easier and more productive because you can concentrate on the function you want to perform and let Hi-Li do all the detail work. The advantage of using the Hi-Li intrinsics is that applications are not affected when system-level changes are made to terminal handling software.

* **System Compatibility:** Hi-Li intrinsics are supported by the HP 3000 MPE/V and the MPE/XL operating systems. Because the procedures are identical on both operating systems, you need learn only one set of procedures, and you do not have to change any code when moving applications between the two systems.

* **Supported Languages:** Hi-Li intrinsics provide high-level language support: you can use Hi-Li with COBOL II, FORTRAN 66/V, FORTRAN 77/V, and Pascal/V.

* **Adding Functions:** Hi-Li intrinsics are designed so that you can add functions to your application by specifying additional parameters, rather than by adding intrinsics and making changes to the code structure.

* **Supported Terminals:** Hi-Li intrinsics support Hewlett-Packard block mode 262X, 239X, and 150 terminals. To make full use of the block mode features, you must use FORMSPEC/V to design your forms. For a complete list of supported terminals and an explanation of how Hi-Li

uses the features of these terminals, see Appendix D. To make full use of the block mode features, you must use FORMSPEC/V to design your forms.

Requirements

Hi-Li intrinsics require the following software and hardware:

* HP 3000 running under MPE/V or MPE/XL;

* HP block mode terminal.

About This Manual

This manual is organized as follows:

Chapter 1:	Provides an overview of the Hi-Li intrinsics and the screen management facility, as well as a brief description of each chapter in the manual.
Chapter 2:	Explains how Hi-Li intrinsics are designed and gives basic rules for using them. Read both Chapter 1 and this chapter before you begin coding with Hi-Li.
Chapter 3:	Describes each of the 12 intrinsics and their parameters in detail. Many of the descriptions include brief examples of how to code parameter structures in each of the supported languages.
Chapter 4:	Outlines each of the six data mapping methods that can be used to map data between the application and fields on the form.
Appendix A:	Provides a complete list of error messages, their meanings, and the action you can take to recover from an error.
Appendix B:	Compares Hi-Li intrinsics and VPLUS/V intrinsics.
Appendix C:	Contains sample application programs for COBOL, Pascal and FORTRAN, as well as a discussion of the Hi-Li diagnostic tracing facility.
Appendix D:	Lists the terminals supported by Hi-Li and explains how Hi-Li uses various terminal features.

Chapter 2 Screen Management Intrinsics

This chapter provides comprehensive information about the Hi-Li intrinsics. If you have never used the screen management facility, read this chapter before you begin using Hi-Li intrinsics to code your application.

The topics in this chapter include:

* The basic structure of Hi-Li intrinsics and how they are used
* The way Hi-Li uses the Application-ready Buffer (ARB)
* The languages that Hi-Li supports
* The data types and structures that Hi-Li uses
* The rules for using Hi-Li

Hi-Li Intrinsics

The HP screen management facility comprises three core intrinsics, HPDSEND, HPDREAD, and HPDPROMPT. They provide the basic functions needed for the exchange of data between the terminal and your application. These core intrinsics are supported by the HPDOPENFORMS, HPDCLOSEFORMS, HPDENABLETERM, and HPDDISABLETERM intrinsics. They perform housekeeping functions such as opening and closing forms files. The remaining intrinsics perform more specialized functions that your application may require, such as configuring parts of the execution environment. Figure 2-1 shows the relationships among the intrinsics.

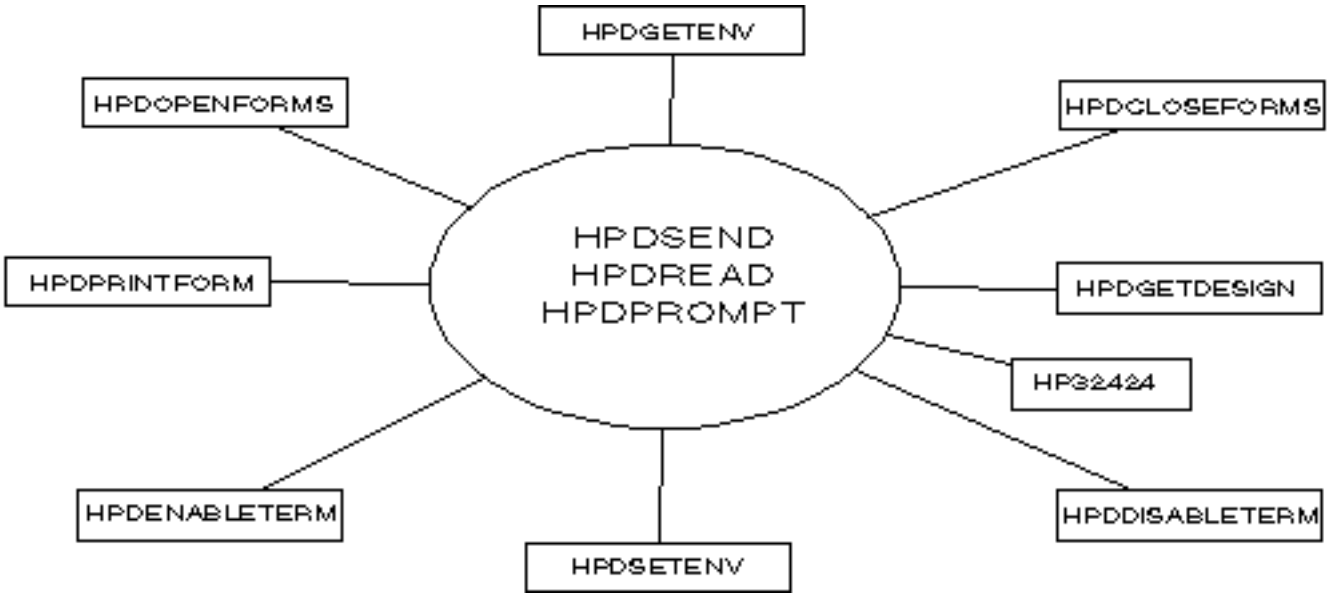

Figure 2-1. Core and Supporting Intrinsics

Core Intrinsics. These intrinsics provide basic functionality.

HPDSEND	Passes application data to the operator's device and arranges it according to a scheme identified or provided by the application.
HPDREAD	Collects data from the operator's device and returns it to the application.
HPDPROMPT	Lets the application focus the operator's attention by altering and highlighting data when it is displayed at the terminal.

Supporting Intrinsics. These intrinsics provide supportive and specialized capabilities.

Supportive Capabilities:

HPDOPENFORMS HPDCLOSEFORMS HPDENABLETERM HPDDISENABLETERM	These intrinsics perform the housekeeping duties of opening and closing forms, and enabling and disabling the terminal.

Specialized Capabilities:

HPDGETENV HPDSETENV	These intrinsics let you see and set the values for options in the execution environment. Use them when your application requires a special configuration.
HPDPRINTFORM	Lets you obtain a printed copy of a form with its data.
HPDGETDESIGN	Lets you retrieve design information to facilitate documenting your application.
HP32424	Lets you identify the current version of the Hi-Li software.

The Hi-Li intrinsics and their parameters are described in detail in Chapter 3.

The Hi-Li Model

Hi-Li is designed to control the transfer of data between the terminal and an application with an interactive user interface. The data is transferred as a block, rather than item by item. The sequence is as follows.

The HPDSEND intrinsic sets up the form and displays it according to

instructions specified in the application. The instructions cover such things as the initialization of field displays, key label display, and the display of application messages in the message window.

The HPDREAD intrinsic reads and collects the information that the operator supplies on the form. If any errors are detected, the HPDPROMPT intrinsic is called to alert the operator to the error by highlighting particular fields and/or displaying messages.

The supporting intrinsics enable the core intrinsics to transfer data by performing the basic housekeeping tasks of enabling the terminal and opening the forms file, and then disabling the terminal and closing the forms file when the application is finished.

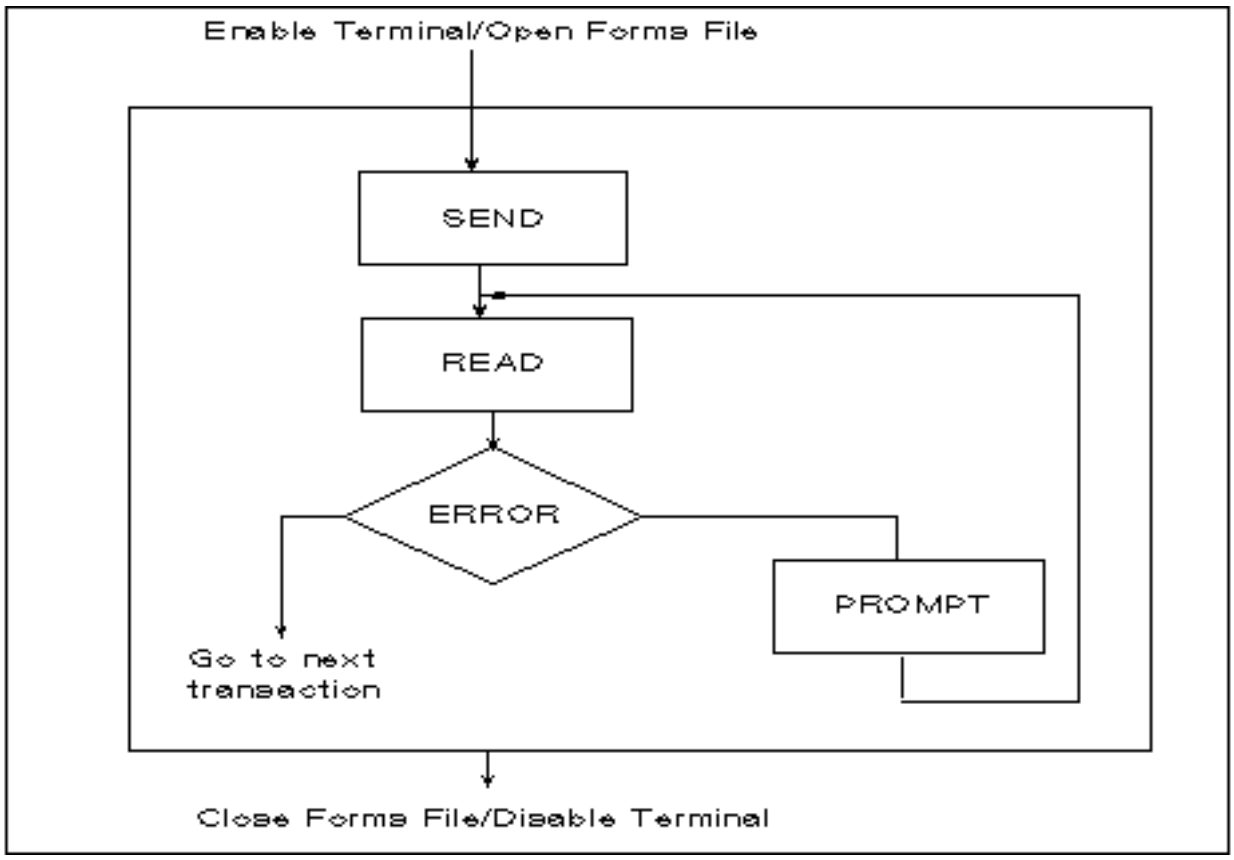

Figure 2-2. The Hi-Li Model

Hi-Li and the Application-ready Buffer (ARB)

A data buffer is passed by the application to HPDSEND, HPDREAD, and HPDPRINTFORM. You can specify the instructions for how you want data moved between this buffer and the form in one of two ways. In the first way, you code your application to specify how you want the contents of the data buffer handled by including a 'data description' parameter. This is the direct method of transferring data, and Hi-Li supports five such methods.

In the second way, you code your application to specify that you want the contents of the data buffer handled according to specifications in the forms file. This is the indirect method, which takes advantage of the Application-ready Buffer (ARB).

Each of these six methods is discussed in detail in Chapter 4.

The Application-ready Buffer. The purpose of the ARB is to represent data as the application expects to use it, and not necessarily as it appears on the screen. For example, a given application may have one arrangement of fields on the screen that corresponds to data entry needs,
and a different internal arrangement that corresponds to file storage layout. In addition, some of the screen data, always in character format, may need to be converted and stored in non-character formats for subsequent processing. Finally, the application's internal buffer may contain data fields that are not for display on the operator's terminal, such as dataset search items (keys) or system checksums.

You can use the ARB to accomplish the data remapping, type conversions, and masking required by your application, instead of having to write application code for these purposes. Use the ARB menus in FORMSPEC/V to specify the necessary data manipulations. Whether or not you choose to use the ARB method for managing data transfers between your application and its associated forms depends on the specific requirements of your application.

For complete information about the Application-ready Buffer, see the *HP Forms Management (VPLUS/V) Reference Manual* (Part No. 32209-90001).

Data Types

Table 2-1 shows the data types and structures that are used in Hi-Li intrinsics.

Table 2-1. Data Types and Structures

FORMSPEC	COBOL	Pascal	FORTRAN
CHAR	X	Packed array CHAR [.._]	CHAR *_
YYDDMM	X(6)	as above	CHAR *6xx
ZONEn	9(_).n	---	---
PACKn	9(_)Vn COMP-3	---	---
SPACKn	S9(-)Vn COMP-3	---	---
REAL	---	REAL	REAL
LONG	---	LONG	Double Precision
INT	S9(4) COMP	Subrange	Integer **2

DINT	S9(9) COMP	-32768..32767 Integer	Integer **4	

Note: [n] represents the number of decimal digits

How Hi-Li Intrinsic Parameters Operate

The Hi-Li intrinsics act as a link between the application and the forms created with FORMSPEC/V. Table 2-2 shows how the information flow is controlled by various parameters as they are passed between Hi-Li and the application.

Table 2-2. Information Flow

Data	How it is Processed
Communications (GLOBALPAK	- Set up by the application - Formatted by *open/enable* - Passed between intrinsics
Status	- Passed back from each intrinsic to the application
Processing Instructions (SENDPAK, READPAK, PROMPTPAK, DATADESCPT, LABELDESCRPT, READITEMS, CURSORPOSITION)	- Passed to each intrinsic by the application
User Interface Specifications (FORMPAK) (FIELDLIST for HPDPROMPT)	- Passed to HPDSEND and HPDPROMPT by the application
User Message (MSG)	- Passed to HPDSEND and HPDPROMPT by the application
Data (DATABUF)	- Passed to HPDSEND by the application - Passed from HPDREAD to the application
Key Labels (LABELBUF)	- Passed to HPDSEND and HPDPROMPT by the application
Edit Report (FIELDLIST from HPDREAD)	Passed from HPDREAD to the application

Abbreviated Parameters. Each intrinsic has multiple parameters. Certain

parameters can be collapsed so that they pass only one part, or pass a null value if they are not necessary to your application. This allows you to build in the feature set you currently need

without having to enter "dummy" information you do not want to include. If you decide to use additional features at a later time, you can simply activate and/or expand the necessary parameter; you do not have to add an additional intrinsic call.

The *chnglisttype* subparameter of the FORMPAK parameter (HPDSEND intrinsic) provides an example of how this works. The *chnglisttype* parameter allows you to indicate whether you want the HPDSEND intrinsic to apply change specifications supplied by the application to fields on a given form. If you assign *chnglisttype* a value of zero (to indicate that the change list is empty), the following five subparameters that detail these change specifications are ignored by Hi-Li.

For example:

changlisttype = 0

then:

listcount
chngentry
fieldident
chngtype
chngspec

are ignored and never referenced by Hi-Li.

Therefore, the FORMPAK parameter can be abbreviated by specifying a *changlisttype* value of zero.

See the HPDSEND intrinsic in Chapter 3 for a complete discussion of the *chnglisttype* parameter and its subparameters.

Supported Languages

Hi-Li intrinsics can be used with COBOL, FORTRAN, and Pascal. The call formats for each of these languages are listed in Table 2-3.

Table 2-3. Language Call Formats

Language	Intrinsic Call Format
COBOL	CALL "intrinsicname" USING parameter1 parameter2 parameter3 . . . parametern.
FORTRAN	CALL intrinsicname (parameter1, parameter2,.|.)
Pascal	intrinsicname (parameter1, parameter2,...);

To ensure consistency among calls from different programming languages, the following rules apply to all parameters:

* Parameters are passed by reference. This means that a literal value cannot be used as a parameter.

* No condition codes are returned. The status of the call is returned in a status word included as part of the RETURNPAK parameter.

* Return type intrinsics are not allowed. Any values returned by the intrinsic are sent as passed parameters.

Hi-Li Dependencies and Rules

The following is a list of rules and dependencies that must be followed and observed when you use the Hi-Li intrinsics.

1. A forms file must be open before you can print the form.

2. A forms file must be open and your terminal must be enabled before you can use the HPDSEND, HPDREAD, and HPDPROMPT intrinsics.

3. The HPDGETENV and HPDSETENV intrinsics have guidelines you must follow that depend on the value you give the MODE parameter. See the HPDGETENV and HPDSETENV intrinsics in Chapter 3 for information about these guidelines.

4. The HPDGETDESIGN intrinsic is not compatible with any of the other Hi-Li intrinsics. This is because the other intrinsics depend on the form referenced by the HPDSEND intrinsic. This form and the form referenced by HPDGETDESIGN may be different. Because of this, you must define a separate GLOBALPAK parameter that you use only with HPDGETDESIGN; you cannot call this intrinsic using the GLOBALPAK parameter that you pass to the other intrinsics.

5. You cannot call tHE HPDREAD and HPDPROMPT intrinsics directly following a call to HPDENABLETERM. This is because Hi-Li expects to see the HPDSEND intrinsic that sets up the form. To call HPDREAD or HPDPROMPT in this case, call HPDSEND first.

6. The following read sequences are allowed:

   a. A read call following a send and/or a prompt.

   b. A read call followed by a second read call with the 'doreread' option.

7. You cannot call HPDREAD or HPDPROMPT directly follow ing a call to HPDPRINTFORM. This is because HPDREAD and HPDPROMPT use the form referenced by the previous HPDSEND intrinsic.

   Only one Hi-Li storage area is allocated for form information. When HPDSEND is called, it sets up this storage area with certain values. When HPDPRINTFORM is called, it resets these values. Therefore, to call HPDREAD or HPDPROMPT after making an HPDPRINTFORM call, call HPDSEND.

8. You cannot mix Hi-Li calls with VPLUS/V calls.

9. A forms file must be opened before you can use the HPDCLOSEFORMS intrinsic to close it.

10. The terminal must be enabled before you can use the HPDDDISABLETERM intrinsic to release it.

11. Consecutive forms file opens are not allowed. Always close the open forms file before opening another.

12. Consecutive terminal enables are not allowed. Always disable the enabled terminal before enabling a second one.

Chapter 3 HI-LI Intrinsics

This chapter describes each of the Hi-Li intrinsics and their parameters,
presented in alphabetical order. Many of the descriptions include an example of how to code various parameter structures of the particular intrinsic in COBOL, Pascal and FORTRAN. Appendix C includes a sample program in each of these languages, illustrating more completely how to use the Hi-Li intrinsics to code an application.

In this chapter, you will find information about:

* The common parameters GLOBALPAK and RETURNPAK
* HPDCLOSEFORMS
* HPDDISABLETERM
* HPDENABLETERM
* HPDGETDESIGN
* HPDGETENV
* HPDOPENFORMS
* HPDPRINTFORM
* HPDPROMPT
* HPDREAD
* HPDSEND
* HPDSETENV
* HP32424

Chapter 3 also includes a summary table of the intrinsics.

Common Parameters

The *globalpak* and *returnpak* parameters are used by each of the intrinsics that make up Hi-Li, except for the HP32424 intrinsic. Because these

parameters are common to most of the intrinsics, a discussion of their uses and their subparameters is given here. Refer to this section when you need information about how to use the *globalpak* and *returnpak* parameters.

GLOBALPAK

Provides the mechanism for information exchange among Hi-Li intrinsics.

Parameter Descriptions

expectedvuf	An eight-byte character array that indicates the revision level of the intrinsics expected by the application. It should be in the format v.uu.ff, and may contain the value A.00.00.
callprotocol	A four-byte integer that indicates how the data passed to the intrinsic is addressed. An identifier specific to the programming language used is assigned: 100 = BASIC/V 110 = HP Business BASIC/V 000 = COBOL II/V 200 = FORTRAN 66/V 210 = HP FORTRAN 77/V 500 = Pascal/V 300 = SPL/V 600 = Transact/V
comarealen	A four-byte integer that indicates the size in bytes of the communications area allocated by the application.
comarea	An array that represents the area reserved by the application for communications between the intrinsics. A minimum of 300 bytes must be allocated for the *comarea*, and this area MUST be initialized to binary zeros before the first intrinsic call.

Discussion

The *globalpak* parameter allows the Hi-Li intrinsics to communicate with one another. Your application should initialize *globalpak* ONCE before the first Hi-Li call; *globalpak* must remain untouched by the application thereafter.

The *comarea* must contain a minimum of 300 bytes. It should be initialized to binary zeros once, after which it must not be used by your application; Hi-Li handles the *comarea* for you.

The *callprotocol* parameter tells Hi-Li the particular programming language you are using. Once set, Hi-Li expects all calls to be in that language's format.

The *expectedvuf* parameter tells Hi-Li what version of its software you

are using. You can initialize this parameter to A.00.00 for this release of Hi-Li unless you want to use the *dontenableinput* option of the HPDPROMPT *promptpak* parameter. If you want to use *dontenableinput*, set this parameter to a value equal to or greater than A.00.10. Once set, this parameter should not be changed. As future enhancements to Hi-Li occur, you will use this parameter to pass the version code for the version containing the features you decide to use.

For information about the *dontenableinput* option of the *promptpak* parameter, see HPDPROMPT in this chapter.

Examples

COBOL:

```
01  globalpak.
    05  expectedvuf       pic x(8).
    05  callprotocol      pics9(8) comp.
    05  comarealen        pic s9(8) comp.
    05  comarea           pic x(300).
```

FORTRAN:

```
CHARACTER*8      EXPECTEDVUF
INTEGER*4        CALLPROTOCOL
INTEGER*4        COMAREALEN
EQUIVALENCE    (GLOBALPAK(1), EXPECTEDVUF),
+              (GLOBALPAK(3), CALLPROTOCOL),
+              (GLOBALPAK(4), COMAREALEN),
+              (GLOBALPAK(5), COMAREA)
```

Pascal:

```
type
  comarea_type     = array [1..75] of integer;

  globalpak_rec = record
    expectedvuf   : packed array [1..8] of char;
    callprotocol  : integer;
    comarealen    : integer;
    comarea       : comarea_type;

 var
   globalpak      : globalpak_rec;
```

RETURNPAK

Returns information about a Hi-Li call to the application.

Parameter Descriptions

returnstatus — A four-byte integer that indicates the status code returned by the intrinsic. The following ranges indicate the status:

0 = successful

<0 = error (programming or system problem)

>0 = exception (operator or data problem)

sublayerstatus — A four-byte integer that indicates the status code returned by the underlying facilities. This information is intended to augment *returnstatus* for diagnostic purposes only.

returnmsglen — A four-byte integer that indicates the length in bytes of *returnmsg*.

returnmsg	A 256-byte character array that represents a displayed message returned by the intrinsic.
lastitemtype	A four-byte integer that indicates what type of item terminated the read intrinsic. A zero (0) means the read intrinsic was terminated by the [ENTER] key or a function key. A one (1) means the read intrinsic was terminated by a field.
lastitemnum	A four-byte integer that indicates the identifier, that is number, of the function key or field that terminated the read intrinsic. For fields, this is the field number FORMSPEC assigns the field. For the [ENTER] key, the value is zero. For function keys, the value corresponds to the function number: f1 = 1, 2 = 2, and so on.
lastitemname	A 32-byte character field that represents the USASCII name of the item that terminated the read intrinsic. What this variable contains depends on the item: **- field** contains a **field name** **ENTER key** shows **$ENTER** **function key** contains **$PFK_n** where n is the key number If the item is not one of the above, *lastitem* contains $VACANT. You can define a string up to 32 characters long to replace the *itemname* returned. For an example of how to do this, see the Discussion for the *returnpak* parameter.
numdataerrs	A four-byte integer that indicates the number of data errors detected during data initialization, editing, or reformatting.
numchngdflds	A four-byte integer returned by HPDREAD that indicates the number of fields into which the operator keyed data. A returned value of -1 means information cannot be determined.

Discussion

The *returnpak* parameter is initialized by the called intrinsic and returns status information when that intrinsic is executed. This status information is only kept until the next intrinsic is executed. For example, if an error message is returned and you call the next intrinsic before responding to the message, the message is lost.

Writing Replacement Strings for LASTITEMNAME

You can choose to replace the name associated with the last item with a string you have defined. You define replacement strings in FORMSPEC by constructing custom error messages that are retrieved by the HPDREAD intrinsic, but not executed. For example, to write a replacement string for the function key f3 within the form ADD_PART, write a processing specification statement for any of the fields in form ADD_PART:

```
IF fieldname NE fieldname THEN
   FAIL "$PFK_3 = replacement string"
```

This statement will never execute because the field will never be unequal to itself. "PFK_3" is the entry key that the HPDREAD intrinsic matches on if the function key f3 terminates the read. All text to the right of the "equals" sign up to 32 characters is copied to *lastitemname*, to be returned to the application.

Examples

COBOL:

```
01   returnpak.
     05  returnstatus     pic s9(8) comp.
     05  sublayerstatus   pic s9(8) comp.
     05  returnmsglen     pic s9(8) comp.
     05  returnmsg        pic x(256).
     05  lastitemtype     pic s9(8) comp.
     05  lastitemnum      pic s9(8) comp.
     05  lastitemname     pic x(32).
     05  numdataerrs      pic s9(8) comp.
     05  numchngflds      pic s9(8) comp.
```

FORTRAN:

```
INTEGER*4        RETURNPAK(79)
INTEGER*4        RETURNSTATUS, SUBLAYERSTATUS,
                               RETURNMSGLEN
CHARACTER*252    RETURNMSG
CHARACTER*4      MSGTAIL
INTEGER*4        LASTITEMTYPE, LASTITEMNUM

CHARACTER*32     LASTITEMNAME
INTEGER*4        NUMDATAERRS, NUMCHNGFLDS
EQUIVALENCE  (RETURNPAK(1), RETURNSTATUS),
+            (RETURNPAK(2),  SUBLAYERSTATUS),
+            (RETURNPAK(3),  RETURNMSGLEN),
+            (RETURNPAK(4),  RETURNMSG),
+            (RETURNPAK(67), MSGTAIL),
+            (RETURNPAK(68), LASTITEMTYPE),
+            (RETURNPAK(69), LASTITEMNUM),
+            (RETURNPAK(70), LASTITEMNAME),
+            (RETURNPAK(78), NUMDATAERRS),
+            (RETURNPAK(79)  NUMCHNGFLDS)
```

Pascal:

```
type
  returnpak_rec = record
  returnstatus    : integer;
  sublayerstatus  : integer;
  returnmsglen    : integer;
  returnmsg       : packed array [1..256] of char;
  lastitemtype    : integer;
  lastitemnum     : integer;
  lastitemname    : packed array [1..32] of char;
  numdataerrs     : integer;
  numchngflds     : integer;
    end;

var
  returnpak       : returnpak_rec;
```

Summary Table

Table 3-1 lists each intrinsic and provides a brief description of its function.

Table 3-1. Summary of the Intrinsics

Intrinsic	Function
HPDCLOSEFORMS	Closes a forms file.
HPDDISABLETERM	Releases a device and optionally reconfigures the device.
HPDENABLETERM	Enables a device and sets up the *comarea* if not already done. It can also optionally return configuration information about the device to the application.
HPDGETDESIGN	Retrieves design information about a specified forms file component.
HPDGETENV	Allows your application to retrieve the configuration of elements in the current execution environment.
HPDOPENFORMS	Opens a forms file and sets up the communication area of the *globalpak* parameter, if not already done.
HPDPRINTFORM	Retrieves and initializes a form, merges data supplied by the application with the form, and outputs both to a spooled printer.
HPDPROMPT	One of the three core intrinsics. It highlights fields, places an application message in the message window, displays key labels, and places the cursor at a selected field.
HPDREAD	One of the three core intrinsics. It reads a device, edits and transforms the returned data according to specifications in the form, and returns the collected information to your application.
HPDSEND	One of the three core intrinsics. It retrieves a form, determines its display position, initializes fields and loads application data, places an application message in the window area, displays the screen and key labels, and places the cursor at a elected input field.
HPDSETENV	Allows your application to configure the elements in the current execution environment.\
HP32424	Identifies the Hi-Li version you have installed.

Each of these intrinsics is discussed in more detail in the following pages.

HPDCLOSEFORMS

Closes a forms file that was opened by HPDOPENFORMS.

Syntax

HPDCLOSEFORMS *globalpak, returnpak, formsfile*

Parameter Descriptions

globalpak	A compound parameter that passes information between intrinsics. For a complete description of this parameter, see "Global Parameters" in Chapter 3.
returnpak	A compound parameter that passes information back from the intrinsic to the application. For a complete description of this parameter, see "Global Parameters" in Chapter 3.
formsfile	An 88-byte character array that indicates the name of the forms file that you want to close. This information is passed to the intrinsic.

Discussion

HPDCLOSEFORMS closes the open forms file. Once closed, this file is no longer available for processing. Call this intrinsic only after the forms file has been opened by HPDOPENFORMS.

HPDDISABLETERM

Releases the device that you enabled using HPDENABLETERM.

Syntax

HPDDISABLETERM *globalpak, returnpak, termpak, devconfig*

Parameter Descriptions

globalpak	A compound parameter that passes information between intrinsics. For a complete description of this parameter, see "Global Parameters" in Chapter 3.
returnpak	A compound parameter that passes information back from the intrinsic to the application. For a complete description of this parameter, see "Global Parameters" in Chapter 3.
termpak	A compound parameter that passes information to the intrinsic.
termfile	An 88-byte character array that indicates the name of the device to be released.
bypassfeature	A four-byte integer that is used by the application to

		indicate the intrinsic features you do not want to use. 0 = ignore 0 = ignore 1 or 3 = do not enable touch sensing 2 or 3 = do not clear device screen
devconfig	A compound parameter that provides in put to and and receives output from the intrinsic.	
	allolen	A four-byte integer that indicates the size in bytes of the allocated configuration storage area.
	actualen	A four-byte integer that indicates, in bytes, the actual length of configuration data. This value is used to re-establish the configuration of the device, and can be set by HPDENABLETERM.
	configarea	An array that represents the area allocated by the application that is used to retain device configuration information. A minimum of 800 bytes is required.

Discussion

HPDDISABLETERM performs a housekeeping role. When the device is released, it is reset back to character mode and closed as a file. You can also request that function key labels and the keys' functions be restored.

HPDENABLETERM

Enables a device.

Syntax

HPDENABLETERM *globalpak, returnpak, termpak, devconfig*

Parameter Descriptions

globalpak	A compound parameter that passes information between intrinsics. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

returnpak	A compound parameter that passes information back from the intrinsic to the application. For a complete description of this parameter, see "Global Parameters" in Chapter 3.	
termpak 5	A compound parameter that passes information to the intrinsic.	
	termfile	An 88-byte character array that indicates the name of the device to be enabled.
	bypassfeature	A four-byte integer that is used by the application to indicate the intrinsic features you do not want to use. 0 = ignore 1 or 3 = do not enable touch sensing 2 or 3 = do not clear device screen
devconfig	A compound parameter that provides input to and and receives output from the intrinsic.	
	allolen	A four-byte integer that indicates the size in bytes of the allocated configuration storage area.
	actualen	A four-byte integer that indicates in bytes the actual configuration data captured.
	configarea	An array that represents the area allocated by the application to retain device configuration information. A minimum of 800 bytes is required.

Discussion

HPDENABLETERM opens the device as a file and places the device in block mode. Once the device is opened and controlled by the Hi-Li intrinsics, you should not call system intrinsics such as FREAD and FWRITE to communicate with the terminal. All device I/O for the opened device must be performed through Hi-Li. If HPDENABLETERM is the first Hi-Li intrinsic called, the *comarea* in *globalpak* is configured.

In addition to opening a device, you can also use this parameter to request that the intrinsic read and return to the application function

key labels and the keys' functions.

NOTE The HPDENABLETERM or HPDOPENFORMS intrinsic must be the first Hi-Li intrinsic you call.

HPDGETDESIGN

Retrieves design information for a particular forms file component.

Syntax

HPDGETDESIGN *globalpak_II, returnpak, mode, keydescrpt, keysbuf, infobuflen, infobuf*

Parameter Descriptions

globalpak_II A compound parameter. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

returnpak A compound parameter that passes information back from the intrinsic to the application. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

mode A four-byte integer that tells the intrinsic the specific forms file component for which you want design information retrieved.

mode	retrieve info for	retrieval key
1	file	forms file
2	form	forms file and form
3	field	forms file, form and field
4	field	forms file, form and field

negative value in this parameter tells HPDGETDESIGN to close the forms file after it completes the retrieval.

keydescrpt An array of four-byte integers that gives the intrinsic a description of the retrieval keys. A zero (0) indicates that the key is in number format and a one (1) indicates that the key is in name format. See the Discussion for this intrinsic for an example of how you might set up the keydescrpt array.

keysbuf A compound parameter that provides a list of retrieval keys for the intrinsic. Each entry in the list corresponds to a retrieval entry in the *keydescrpt* parameter. Retrieval keys that are specified in name format are 32-byte character arrays (except for file names, which are 88 bytes). They contain a USASCII component name that is left justified.

Retrieval keys that are specified in number format are four-byte integers. If the value is

	positive, the number is the one assigned the component by the design facility. If the value is negative, the number indicates the order of the component within the design.
infobuflen	A four-byte integer that tells the intrinsic the length in bytes of the *infobuf* parameter.
infobuf	The area into which the information you have requested is returned from the intrinsic. The format of the information returned is specific to the type of design component you requested: file, form, or field. Format is also determined by the design facility that was used to create the forms. See Table 3-2 for complete information about how information and formats are returned from the intrinsic. The amount of information returned is determined by either the applicable format or the contents of *infobuflen*, whichever is less.

Discussion

HPDGETDESIGN is a special-purpose intrinsic that lets you extract design information about a FORMSPEC/V forms file. Use this intrinsic to document forms files or to generate source code data structures such as copylibs.

Because HPDGETDESIGN is a special-purpose intrinsic, you must allocate a separate copy of the *globalpak* parameter to pass to the intrinsic. You cannot pass this copy of *globalpak* to any of the other Hi-Li intrinsics.

For efficiency, the intrinsic opens the forms file for you. If you want to close the file, use a negative number for the value of the *mode* parameter.

The *keydescrpt* parameter tells HPDGETDESIGN the format, number or name of the retrieval key. For example, if you are retrieving information about a field, you might set up the *keydescrpt* array in your application as follows:

* Assign 1 to the first integer in the array to indicate the first key (file) is in name format.

* Assign 0 to the second integer in the array to indicate the second key (form) is in number format.

* Assign 1 to the third integer in the array to indicate the third key (field) is in name format.

Table 3-2 lists the *infobuf* subparameters and shows how information and formats of FORMSPEC/V forms files are returned from the intrinsic. The value of *mode* is treated as an absolute number.

1. File Information Retrieval (*mode* = 1)

Table 3-2. INFOBUF Formats

fileversion	four-byte integer *infobuflen* = 4 File version number that is a data/time stamp. It is recorded when the forms file was last compiled.
numforms	four-byte integer *infobuflen* = 8 Number of forms in the file.
maxnumflds	four-byte integer *infobuflen* = 12 Maximum number of fields in any one form.
maxdatabuflen *	four-byte integer *infobuflen* = 16 Maximum data buffer size in bytes. It may be overstated by two bytes.
*numsavefld*s	four-byte integer *infobuflen* = 20 Number of save fields in file.
headform	32-byte character array *infobuflen* = 52 Name of head form in file.
globerrenh	8-byte character array *infobuflen* = 60 Global error enhancement. This is a combination of I, H, U, B, 1-8, or NONE.
globwindowenh	8-byte character array *infobuflen* = 68 Global window enhancement. This is a combination of I, H, U, B, 1-8, or NONE.
windowposition	four-byte integer *infobuflen* = 72 The line on the screen where the window appears. This is line number 0-24. The line closest to the top of the form is 1. Zero (0) means no window.

2. Form Information Retrieval (*mode* = 2)

formname	32-byte character array *infobuflen* = 32 Form name.
formnum	four-byte integer *infobuflen* = 36 Form number. This number represents the order of the form within the file and may change if the forms file is recompiled.
numflds	four-byte integer *infobuflen* = 40 The number of fields in the form.
databuflen *	four-byte integer *infobuflen* = 44 The data buffer length in bytes.
nextform	32-byte character array *infobuflen* = 76 The name of the next form.
repeatopt	four-byte character array *infobuflen* = 80 Repeat option. It can be N, A, or R.
nextformopt	four-byte character array *infobuflen* = 84 Next form option. It can be C, A, or F.

Field Information Retrieval (*mode* = 3)

formname	32-byte character array *infobuflen* = 32 The name of a form.
fldname	32-byte character array *infobuflen* = 64 Field name.
scrnorder	four-byte integer *infobuflen* = 68 The field number according to its order on the screen.
fldnum	four-byte integer

	infobuflen = 72
	The field number according to the order in which it was created.
fldlen *	four-byte integer
	infobuflen = 76
	Field length in bytes.
databufoffst	four-byte integer
	infobuflen = 80
	Position of the field in the data buffer, offset from one.
fldenh	eight-byte character array
	infobuflen = 88
	Field enhancement. It is a combination of I, H, U, B, S, 1-8, or NONE.
datatype	four-byte character array
	infobuflen = 92
	Data type of field. This can be CHAR, DIG, IMPn, MDY, DMY, YMD, or NUM(n)
fldtype four-byte character array *infobuflen* = 96	
Field type. It can be O, R, P, or D.	

4. Field Information Retrieval (*mode* = 4)

fieldinitstring	A character array equal to the field length or the length defined in *infobuflen*, whichever is less. This array returns a character string equal to the field's initial value.

* = size in characters (bytes)

When you request a format, you get all the formats of a length up to and including the length you specify in *infobuflen*. For example, if you request a form retrieval and pass an *infobuflen* of 16, you see information for the first four parameters; *fileversion, numforms, maxnumflds*, and *maxdatabuflen*.

HPDGETENV

Retrieves configuration elements of the current execution environment.

Syntax

HPDGETENV *globalpak, returnpak, mode, envbufdescrpt, envbuf*

Parameter Descriptions

globalpak — A compound parameter that passes information between intrinsics. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

returnpak — A compound parameter that passes information back from the intrinsic to the application. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

mode — A four-byte integer that identifies the element for which you are receiving current configuration information. You can use the following values:

1 = interface library version

2 = forms file native language identifier

3 = auto test option

4 = device token

5 = form token

envbufdescrpt — A four-byte integer. It is used in those cases where your application is allowed to specify the amount of information that is to be retrieved and how that information is to be arranged within the *envbuf* parameter.

Note: The output format of the most of the available configuration retrievals is fixed, in which case this parameter is ignored.

envbuf — A compound parameter passed from the intrinsic. This is the area into which the requested element configuration is returned. The data structure you supply for this parameter depends on the mode you have specified:

(If) Mode	Element	(Supply) Length/Format
1	Library version	An eight-byte character array
2	Native language	A four-byte integer. The forms file must be open to use this mode. See the *Native Language Support Reference Manual* for the applicable codes.
3	Auto test option	A four-byte integer. The forms file must have been opened or the device enabled to use this mode. The following values are returned: 0 = disabled 1 = enabled
4	Device token	An array of 10 four-byte integers. If the device is not enabled, the information returned is undefined. The array is defined as follows: The first integer in the array contains the file number assigned to the open device. A zero is returned if the device is not open. The second integer in the array contains the system logical device number assigned to the open device. A zero is returned if the device is not open. The third integer in the array contains the device type identifier of the open device. A zero is returned if the device is not open. See Table D-1 in Appendix D for a list of device type identifiers that you can specify. The remainder of the array is undefined.
5	Form token	A 200-byte record defined as:
	currentformname	A 32-byte character array that contains the name of the form currently being processed.
	nextformname	A 32-byte character array that contains the name of the next form to be processed.
	currentformopt	A four-byte integer that indicates whether the current form is a repeating form, and if so, whether it is to be appended to itself. The values returned are: 0 = normal sequence, do not repeat or append 1 = repeat current form 2 = repeat current form and append it to itself
	nextformopt	A four-byte integer that indicates whether the screen is to be cleared when the next form is displayed or whether the next form is to be appended to the current form. It also indicates whether the form, if appended, is to be frozen on the screen. The values returned are: 0 = clear screen, do not freeze or append 1 = append next form to current form 2 = freeze current form and append next form to it.
	databuflen	A four-byte integer that indicates the data buffer length, in bytes, of the current form.
	screenlen	A four-byte integer that indicates the screen length, in lines, of the current form.
		The remainder of the record is undefined.

The forms file must be open to use Mode 5.

Discussion

HPDGETENV is a special-purpose intrinsic meant for advanced programming applications. It allows your application to retrieve information about the current environment. You can request information about the environment for the following five elements:

1. **Library version -** This allows your application to determine which version of Hi-Li software is installed.

2. **Native language identification -** This allows your application to retrieve the NLS code assigned to the open forms file.

3. **Auto Test -** This allows you to determine if terminal interaction is executing in auto test mode.

4. **Device token -** This allows you to receive particular information about the open terminal. If the terminal you are querying is not enabled, the information returned is undefined.

5. **Form token -** This allows you to receive particular information about the current form.

HPDOPENFORMS

Opens a forms file.

Syntax

HPDOPENFORMS *globalpak, returnpak, formsfile*

Parameter Descriptions

globalpak	A compound parameter that passes information between intrinsics. For a complete description of this parameter, see "Global Parameters" in Chapter 3.
returnpak	A compound parameter that passes information back from the intrinsic to the application. For a complete description of this parameter, see "Global Parameters" in Chapter 3.
formsfile	An 88-byte character array that indicates the name of the forms file you want to open. This information is passed to the intrinsic.

Discussion

HPDOPENFORMS opens the forms file and sets up the communications area within the *globalpak* parameter, if it has not already been done.

HPDPRINTFORM

Integrates text with a form and prints the result.

Syntax

HPDPRINTFORM *globalpak, returnpak, formcntrl, fillcntrl, devicenum, datadescrpt, databuf*

Parameter Descriptions

globalpak — A compound parameter that passes information between intrinsics. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

returnpak — A compound parameter that passes information back from the intrinsic to the application. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

formcntrl — A compound parameter that provides printing information to the intrinsic. It is made up of the following subparameters:

formname — A 32-byte character array that supplies the USASCII character name of a form. This name represents the scheme used to arrange data that is sent to the printer.

undrlncntrl — A four-byte integer to specify whether or not to underline fields. The values you can use are:

0 = do not underline fields
1 = underline all fields

pagecntrl — A four-byte integer that specifies page control. The values you can use are:

0 = page eject
1 = carriage return/line feed

Note: Page control works before a form is printed; that is, "advance before writing".\

enablereformat — A four-byte integer used by the application to indicate that the data is to be reformatted after it is merged with the form. It is

reformatted before the form and data are printed according to the specifications embedded in the form. You can use the following values:
0 = ignore reformatting specifications
1 = reformat data

fillcntrl A compound parameter that tells HPDPRINTFORM to replace, or fill, blanks within fields when printing the form. The subparameters and the functions they represent are:

filldesc A four-byte integer that specifies the degree of functionality. The values you can specify are:
0 = no fill
1 = fill trailing blanks
2 = fill leading blanks
3 = fill trailing and leading blanks,newline>

entrycnt A four-byte integer that specifies the number of fields to be filled.

fldidtype A four-byte integer that specifies the format of field identifiers, numbers or names. The values you can specify are:

0 = number format
1 = name format

entrydesc Identifies a table defined in your application.

fldid If you have given *fldidtype* the value 0, this is a four-byte integer that identifies by field number or field order the field to be filled. The values you can specify are:
> 0 = field number
< 0 = field order

If you have given *fldidtype* the value 1, this is a 32-byte character array that contains the USASCII name of the field to be filled.

fillchar A four-byte character array that identifies the fill character. Blanks may be used as a fill character. The array is set up as follows:

The first byte contains the fill character.

Bytes two through four are reserved for system use.

Note: You cannot fill a field larger than 2000 bytes. If you attempt to fill a field that contains more than 2000 bytes, a warning is

	generated telling you that only the first 2000 bytes of the field were filled.
devicenum	A four-byte integer providing input to and output from HPDPRINTFORM, that specifies the file number of the list file to which the form will be written. If you specify a zero, HPDPRINTFORM opens a list file named FORMLIST with a device class of LP. This is equivalent to the file equation FILE FORMLIST; DEV=LP; REC=-80 Because this parameter is used for both input and output from the parameter, your application can interleave calls to this intrinsic with its own writes to the list file. The Discussion for this intrinsic provides an example.
datadescrpt	A compound parameter that tells the intrinsic how data is to be mapped when moving it between the application and the data fields on a form. There are six data mapping methods. The first and simplest uses the following values: -1 = do not transfer data 0 = move all data. Data is moved as a concatenated string of data from the *databuf* parameter as type CHAR. The number of bytes moved is equal to the sum of the lengths of all the fields on the form. See Chapter 4 for information about the other five data mapping methods you can specify.
databuf	A record that defines for HPDPRINTFORM the application area from where data is copied to the fields on a form.

Discussion

HPDPRINTFORM gets and initializes a form, merges data supplied by the application with the initialized form, finishes the form, and sends the form and data to a spooled printer as output. The forms file must be open before you call this intrinsic. You do not need a terminal to use this intrinsic. An example of how you might use this intrinsic is to print tickets.

As mentioned under the *devicenum* parameter, your application can mix application writes with form prints, allowing your application to interleave data from a form with other information. An example of this is printing shipping receipts. Using the HPDPRINTFORM intrinsic, the application prints a receipt using a predefined form. Then, depending on
the material being received, special handling instructions can be print-ed
to accompany the receipt.

HPDPROMPT

Prompts the operator.

Syntax

HPDPROMPT *globalpak, returnpak, promptpak, cursorposition, msg, fieldlist, labeldescrpt, labelbuf*

Parameter Descriptions

globalpak — A compound parameter that passes information between intrinsics. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

returnpak — A compound parameter that passes information back from the intrinsic to the application. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

promptpak — A compound parameter that modifies the default operations of HPDPROMPT. It has the following subparameters:

repaintdata — A four-byte integer that tells HPDPROMPT to force a rewrite of the operator's data display. Use this parameter to keep displayed data consistent with the data in internal buffers. You can supply the following values:

0 = ignore
1 = force a rewrite of the data to the device

windowenh — An eight-byte character array that updates the window line enhancement. If the array contains all spaces or binary zeros, the current window enhancement is used. If the array contains "NONE" left justified, the window is displayed without enhancement. If you do not supply either of these values for the array, you can supply any of the following values in any combination:

I = inverse video
H = half bright - this has no effect unless inverse video is set on.

U = underlined
B = blinking

resethilited A four-byte integer that tells HPDPROMPT whether or not to reset those fields that are highlighted due to errors. You can supply the following values:

0 = ignore the fields flagged in error
1 = reset the flagged fields

dontenableinput A four-byte integer indicating that if the keyboard is locked, it should remain locked; it does not do an actual lock. You can supply the following values:
0 = ignore
1 = do not enable device to take operator input.

Use this option if, for example, you have an application that displays progress messages during a long transaction. This subparameter locks the keyboard to prevent the operator from inadvertently making a keyboard entry.

Note: This subparameter is only recognized if you have supplied a value equal to or greater than A.00.10 for the *globalpak* subparameter, *expectedvuf*.

bypassfeature A four-byte integer that will be used for future Hi-Li implementations. At that time, you must supply the *globalpak* subparameter, *expectedvuf*, with a value equal to or greater than A.00.10. For this release of Hi-Li software, set *bypassfeature* to zero.

cursorposition A four-byte integer or a 36-byte array of characters that identifies the field at which

the operator is to begin entering data. It also tells HPDPROMPT to position the cursor at this field. The field must be an input field.

A four-byte integer identifies the beginning field by number or screen order. A positive integer indicates a field number. A negative integer indicates screen order. If you supply a zero, HPDPROMPT moves the cursor to the first enhanced input field on the screen. If no fields are enhanced, the cursor is positioned at the first physical input field on the screen.

OR

A 36-byte character array that identifies the beginning field by its USASCII name. This array must be preceded by a double backslash (\\). If you pass spaces in this array, HPDPROMPT moves the cursor to the first enhanced input field on the screen. If no fields are enhanced, the cursor is positioned at the first physical input field on the screen.

msg A compound parameter that tells HPDPROMPT the message that you want displayed in the message window, and the length of that message. It is made up of the following subparameters:

msglen A four-byte integer that indicates in bytes the length of the message you want HPDPROMPT to transfer to the window area. If you supply a zero, HPDPROMPT fills the window with blanks. If you supply a negative number, HPDPROMPT does not update the window area and the previous message is displayed.

msgline A character array of up to 256 bytes that identifies the message you want HPDPROMPT to display in the window area.

fieldlist A compound parameter that identifies for HPDPROMPT the fields you want highlighted. This parameter is compatible with the HPDREAD fieldlist parameter. It is made up of the following subparameters:

listtype A four-byte integer that tells HPDPROMPT to list the fields by number or name. You can supply the following values:
0 = null list
1 or 2 = fields listed in number format

		3 = fields listed in name format
	allocnt	A four-byte integer that is not used by HPDPROMPT.
	actualcnt	A four-byte integer that indicates the number of active entries in the list of fields.
	listentry	Identifies the table of fields defined in your application.
	fieldident	If you supplied a value of 1 or 2 for *listtype*, this is a four-byte integer that identifies a field. A positive number indicates field number. A negative number indicates screen order. If you supplied a value of 3 for *listtype*, this is a 32-byte character array that identifies the USASCII character name of a field.
labeldescrpt	A compound parameter that identifies for HPDPROMPT the key labels that your application will update. Function keys are labeled, by default, with the label strings defined in the forms file. *labeldescrpt* is made up of the following subparameters:	
	descrptcnt	A four-byte integer that indicates the number of labels to be updated.
	descrptentry	Identifies a table defined in your application.
	labelident	A four-byte integer that identifies the key label number.
	labelenh	An eight-byte character array that will be used by a future Hi-Li enhancement to tell HPDPROMPT to accept whatever the current enhancement is. It is not currently used by HPDPROMPT. It is recommended that you set this variable to spaces or binary zeros.

OR

labeldescrpt is made up of the following set of subparameters:

labelset	A 36-byte character array specified by double backslashes followed by the USASCII character label set name. Specifying (double backslash) $RESET_FORM tells Hi-Li to reset function key labels to the base set specified for the current form. If a label set was not defined for the form, Hi-Li uses the forms file global set as the base set. Specifying spaces tells Hi-Li not to update the current function key labels. For a discussion of the *labelset* subparameter, see the Discussion for HPDPROMPT that follows these parameters.

labelbuf A byte array of characters that identifies for HPDPROMPT the application area that contains the label strings. For the HP 3000, 16 bytes are sent to each label.

Discussion

HPDPROMPT is one of the three core intrinsics that make up the screen management intrinsics. Use this intrinsic to display messages, highlight and position the cursor at selected fields, and update key labels when an operator error is detected. Parameters that are not required by your application can be compacted and passed as abbreviated parameters.

<3>Creating Function Key Label Sets Using LABELSET

Because there is no facility within FORMSPEC/V for defining free-floating function key label sets, the HPDPROMPT and HPDSEND *labelset* subparameter (of the *labeldescrpt* parameter) provides this ability by "borrowing" the function key label set from the form you identify in the *labelset* subparameter.

The form you identify can be any of the "live" forms in the forms file or it can be a "dummy" form which exists only as a function key label set place holder. You can create a place holder for a key label set by

adding a form to the forms file, defining a set of labels for the form, and building a one character, text only, screen image for the dummy form.

Examples

COBOL:

```
01   fieldlist.
     05  listtype              pic s9(8) comp.
     05  allocnt               pic s9(8) comp.
     05  actualcnt             pic s9(8) comp.
     05  listentry  occurs 5 times.
         10 fieldid             pic s9(8) comp.

01   labeldescrpt.
     05  descrptcnt            pic s9(8) comp.
     05  descrptentry  occurs 8 times.
         10  labelident        pic s9(8) comp.
         10  labelenh          pic x(8).
```

FORTRAN:

```
INTEGER*4        FIELDLIST(8)
INTEGER*4        LISTTYPE
INTEGER*4        ALLOCNT
INTEGER*4        ACTUALCNT
EQUIVALENCE (FIELDLIST(1), LISTTYPE),
+           (FIELDLIST(2), ALLOCNT),
+           (FIELDLIST(3), ACTUALCNT)
    INTEGER*4    FIELDID(1,5)
    EQUIVALENCE (FIELDLIST(4), FIELDID)
```

For an example of how a record structure can be manipulated in FORTRAN, see the COLLECT_TXNS subroutine in the FORTRAN example program in Appendix C.

Pascal:

```
type
  fieldlist_rec = record
    listtype          : integer;
    allocnt           : integer;
    actualcnt         : integer;
    fieldid           : array [1..5] of integer;
      end;

var
  fieldlist         : fieldlist_rec;
```

HPDREAD

Reads, edits, and transforms data, and returns the information to the application.

Syntax

HPDREAD *globalpak, returnpak, readpak, readitems, cursorposition, datadescrpt, databuf, fieldlist*

Parameter Descriptions

globalpak — A compound parameter that passes information between intrinsics. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

returnpak — A compound parameter that passes information back from the intrinsic to the application. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

readpak — A compound parameter that modifies the default operations of HPDREAD. It is made up of the following subparameters:

- *readtime* — A four-byte integer that tells HPDREAD how many seconds to allow for a read. The values you can use are:
  0 = do not time the read
  >0 = time the read and report an exception if a timeout occurs
- *enablereformat* — A four-byte integer used by the application to tell HPDREAD to reformat the data after it is collected and edited, and before it is returned to your application. Data is reformatted according to the specifications embedded in the form. Reformatting does not include data type conversions. You can use the following values:
  0 = ignore, do not reformat data
  1 = reformat data
- *doreread* — A four-byte integer that is used by the application to indicate that this read is a consecutive or follow-up read. You can supply the

following values:

0 = ignore
1 = automatically read the terminal. The read termination item will always be identified as the ENTER key.

bypassfeature A four-byte integer that will be used for future Hi-Li implementations. At that time, you must supply the *globalpak* subparameter, *expectedvuf*, with a value equal to or greater than A.00.10. For this release of Hi-Li software, set *bypassfeature* to zero.

readitems A compound parameter used by the application to define valid read termination items, (keys and fields), and the tasks HPDREAD is to perform following the termination of the physical read. This information is used as input to HPDREAD. It is made up of the following subparameters:

itemcnt A four-byte integer that indicates the number of active termination item entries. Setting *itemcnt* to zero tells HPDREAD to use implicit read key definitions, that is, read, edit, and report errors on key 0, and interrupt only on function keys 1 through n and fields 1 through n. This is the same as options 2 and 0, respectively, of the *readopt* subparameter discussed later under the HPDREAD intrinsic.

itementry Identifies a table defined in your application that contains termination item types, termination items, and read options.

itemtype A four-byte integer that indicates the termination item type, a key or field. The values you can use are:
0 = key

1 = field

itemident A four-byte integer that identifies a termination item. If you have defined *itemtype* as a key, the ENTER key is 0, function key 1 is 1, and so on.

If you have defined *itemtype* as field, a positive number indicates a field number and a negative number indicates screen order.

readopt A four-byte integer that tells HPDREAD which read option you want to use. You can use the following values:

-1 = no read

This option returns an exception to the application, and is the implicit entry of all items (keys and fields) that are not defined in the table identified by *itementry*.
0 = interrupt only
1 = unedited read
2 = read and edit - report errors OR return data
3 = read and edit - report errors AND return data

Note: If you specify option 3 and data editing errors occur, you will get an error when type conversions are performed. Use option 3 when you are not doing type conversions.

cursorposition A four-byte integer or 36-byte character array that is reserved for a feature not yet implemented. Set this parameter to zero or spaces.

datadescrpt A compound parameter that tells HPDREAD how data is to be mapped when it is moved between data fields on the form and your application. There are six ways to map data. The first and simplest way is shown below.
-1 = do not transfer data
0 = moves concatenated string of data to *databuf*.

The number of bytes moved is equal to the sum

of the lengths of all the fields in the form. All data is moved as type CHAR. See Chapter 4 for a description of each of the other five mapping methods.

databuf A record that represents the application area that receives the data copied from fields on the form. It is passed as output from the intrinsic.

fieldlist A compound parameter that allows your application to get a list of the fields that are flagged in error. This parameter is both input to and output from HPDREAD. It is compatible with the HPDPROMPT fieldlist parameter, and is made up of the following subparameters:

listtype A four-byte integer used by your application to indicate how HPDREAD should identify fields in the list. You can use the following values:
0 = do not generate a list
1 = identify by field number
2 = identify by screen order
3 = identify by field name

allocnt A four-byte integer used by your application to indicate the number of entries that are allocated for the returned list.

actualcnt A four-byte integer that is the number of defined entries in the list. This value is set by HPDREAD. If no fields are flagged in error, and the value of the *allocnt* parameter is greater than zero, HPDREAD sets *actualcnt* to zero.

listentry Identifies a table defined in your application that will receive the list of fields in error.

Note: This table is sorted by field number in the order the fields were created. This is true regardless of the field ID method you use (*listtype*).

fieldident	If you supplied a value of 1 or 2 for *listtype*, this is a four-byte integer. If you supplied 1, fields are identified by number. If you supplied 2, fields are identified by screen order. If you supplied a value of 3 for *listtype*, this is a 32-byte character array. Fields are identified with a USASCII character name.

Discussion

HPDREAD is one of the three core intrinsics that make up the screen management intrinsics. It lets you read a device, such as a terminal, and edit and reformat the data according to the processing specifications in the form. It then returns this information to your application. Parameters that are not required by your application can be compacted and passed as abbreviated parameters.

Examples

COBOL:

```
01  readitems.
    05  itemcnt              pic s9(8) comp.
    05  itementry   occurs 9 times.
        10  itemtype         pic s9(8) comp.
        10  itemident        pic s9(8) comp.
        10  readopt          pic s9(8) comp.

01  fieldlist
    05  listtype             pic s9(8) comp.
    05  allocnt              pic s9(8) comp.
    05  actualcnt            pic s9(8) comp.
    05  listentry   occurs 5 times.
        10  fieldid          pic s9(8) comp.
```

FORTRAN:

```
INTEGER*4        READITEMS(28)
INTEGER*4        ITEMCNT
EQUIVALENCE      (READITEMS(1), ITEMCNT)
   INTEGER*4        ITEMTYPE(3,9)
   INTEGER*4        ITEMIDENT(3,9)
   INTEGER*4        READOPT(3,9)
   EQUIVALENCE      (READITEMS(2), ITEMTYPE),
+                   (READITEMS(2), ITEMIDENT),
+                   (READITEMS(2),READOPT)
```

For an example of how a record structure can be manipulated in FORTRAN, see the COLLECT_TXNS subroutine in the FORTRAN example program in Appendix C.

```
INTEGER*4        FIELDLIST(8)
INTEGER*4        LISTTYPE
INTEGER*4        ALLOCNT
```

```
      INTEGER*4        ACTUALCNT
      EQUIVALENCE (FIELDLIST(1), LISTTYPE),
     +            (FIELDLIST(2), ALLOCNT),
     +            (FIELDLIST(3), ACTUALCNT)*j*Q
          EQUIVALENCE (FIELDLIST(4), FIELDID)
```

For an example of how a record structure can be manipulated in FORTRAN, see the COLLECT_TXNS subroutine in the FORTRAN example program in Appendix C.

Pascal:

```
type
  itementry_rec = record
    itemtype          : integer;
    itemident         : integer;
    readopt           : integer;
      end;

readitems_rec = record
  itemcnt           : integer;
  itementry         : array [1..9] of
                      itementry_rec;
    end;

var
  readitems         : readitems_rec;

type
  fieldlist_rec = record
    listtype          : integer;
    allocnt           : integer;
    actualcnt         : integer;
    fieldid           : array [1..5] of integer;
      end;

var
  fieldlist          : fieldlist_rec;
```

HPDSEND

Gets a form and displays it on the screen.

Syntax

HPDSEND *globalpak, returnpak, sendpak, formpak, cursorposition, msg, datadescrpt, databuf, labeldescrpt, labelbuf*

Parameter Descriptions

globalpak — A compound parameter that passes information between intrinsics. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

returnpak — A compound parameter that passes information back from the intrinsic to the application. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

sendpak — A compound parameter that modifies the default operations of HPDSEND. It is made up of the following subparameters:

dontenableinput — A four-byte integer that is used by the application to

	indicate that you want to make consecutive calls to HPDSEND. Use this parameter if, for example, your application displays a header, and then appends a detail form. This insures that no data can be entered until the entire screen is displayed, eliminating the possibility of corrupting the screen display. The values you can specify are: 0 = ignore 1 = do not enable device to take operator's input; keeps a locked device from being unlocked.
windowenh	An eight-byte character array that updates the window line enhancement. If the array contains all spaces or binary zeros, the current window enhancement is used. If the array contains "NONE" left justified, the window is displayed without enhancement. If you do not supply either of these values for the array, you can specify the following values in any combination: I = inverse video H = half bright - this has no effect unless inverse video is on. U = underlined B = blinking
bypassfeature	A four-byte integer used by the application to indicate the features you do not want HPDSEND to use. The values you can specify are: 0 = ignore

1 or 3 = do not initialize fields with form specifications.

2 or 3 = do not force the operator's data display to be refreshed.

formpak A compound parameter that acts as input to HPDSEND. It provides HPDSEND with specific information about displaying forms on the operator's terminal screen. It is made up of the following subparameters:

formname A 32-byte character array that names the form to be displayed on the user's device. You can alternatively specify one of two formname tokens: **$REFRESH** or **$PREDETERMINED**.

$REFRESH is used to reinitialize the operator's terminal. It causes the following actions to be performed:

- reset and reinitialize the terminal

- redisplay the last form at the home position (top left corner of the screen)

- redisplay field data and enhancements

- redisplay window line contents and enhancement

Use the **$REFRESH** token to implement a recovery mechanism for terminal power failures.

$PREDETERMINED tells HPDSEND that your application will use the form and form positioning directive that are specified in the forms file. Alternately, the current form can contain embedded processing instructions that, when executed, determine which form to use next and where to position it.

formposition A four-byte integer that tells HPDSEND where to position the form on the operator's terminal screen. The values you can specify are:

0 = home - clears the device and positions the form at the top left of the screen

1 = overlay - position at the beginning of last form.

2 = append - position at end of last form

	3 = freeze and append - position at the end of the last form after the last form is frozen.
chnglisttype	A four-byte integer that is input to HPDSEND. It tells HPDSEND whether or not to apply application supplied change specifications to fields on the form. The values you can specify are: 0 = field change list is empty 1 = fields to change are identified by number 2 = fields to change are identified by name.
listcount	A four-byte integer that is input to HPDSEND and tells it the number of field change entries in the list.
chngentry	Identifies the table of entries defined in your application.
fieldident	If you gave *chnglisttype* a value of 1, this four-byte integer is input to HPDSEND. A positive value indicates field number. A negative value indicates screen order. If you gave *chnglisttype* a value of 2, this 32-byte character array is input to HPDSEND. It is the USASCII character name of a field.
chngtype	A four-byte integer that provides input to HPDSEND about field attributes to be changed. The values you can specify are: -1 = do not change field attributes 1 = toggle enhancement attribute

2 = toggle field type attribute

3 = toggle data type attribute

4 = change enhancement attribute

5 = change field type attribute

6 = change data type attribute

7 = toggle field error enhancement attribute. This does not return the enhancement to the application.

8 = change to field error enhancement attribute

Note: For values 1, 2, and 3, toggle sets the attributes to the new value and returns the old value to the application.

chngspec An eight-byte character array that provides input to and output from HPDSEND. The *chngspec* values allowed depend on the value you give *chngtype*:

If *chngtype* has a value of 1 or 4, *chngspec* values NONE or any combination of H, I, B, and U are allowed.

If *chngtype* has a value of 2 or 5, *chngspec* values O, D, P, and R are allowed.

If *chngtype* has a value of 3 or 6, *chngspec* values CHAR, DIG, IMPn, NUM(n), DMY, MDY, and YMD are allowed.

If *chngtype* has a value of 1, 2, or 3 (toggle specifications), the field specification that was in effect before the change took place is returned to the application.

If *chngtype* has a value of 7 or 8, the content of *chngspec* is ignored.

See "Field Menu" in the *VPLUS/V Reference Manual* for definitions of the above enhancements, field types, and data types.

cursorposition A four-byte integer or a 36-byte array of characters that identifies the field at which the operator is to begin entering data. It also tells HPDSEND to position the cursor at this field. The field must be an input field.

A four-byte integer identifies the beginning field by number or screen order. A positive integer indicates a field number. A negative integer indicates screen order. If you supply a zero, HPDSEND moves the cursor to the first input field on the screen.

A 36-byte character array identifies the beginning field by its USASCII name. This array must be preceded by a double backslash. If you pass spaces in this array, HPDSEND moves the cursor to the first input field on the screen.

msg A compound parameter that tells HPDSEND the message you want displayed in the message window, and the length of that message. It is made up of the following subparameters:

msglen A four-byte integer that indicates in bytes the length of the message you want HPDSEND to transfer to the window area. If you supply a zero, HPDSEND fills the window with blanks. If you supply a negative number, HPDSEND does not update the window area and the previous message is displayed.

msgline A character array of up to 256 bytes that identifies the message you want HPDSEND to display in the window area.

datadescrpt A compound parameter that tells HPDSEND how data is to be mapped when it is moved between your application and data fields in the form. There are six ways to map data. The first and simplest way is shown below.

-1 = do not transfer data

0 = moves concatenated string of data from *databuf*. The number of bytes moved is equal to the sum of the lengths of all the fields in the form. All data is moved as type CHAR.

See Chapter 4 for a description of each of the

other five mapping methods.

databuf A record passed as input to HPDSEND that represents the application area from which the data is copied to fields on the form.

labeldescrpt A compound parameter that identifies for HPDSEND the key labels that your application will update. Function keys are labeled, by default, with the label strings defined in the forms file. *labeldescrpt* is made up of the following subparameters:

descrptcnt A four-byte integer that indicates the number of labels to be updated.

descrptentry Identifies a table defined in your application.

labelident A four-byte integer that identifies the key label number.

labelenh An eight-byte character array that will be used by a future Hi-Li enhancement to tell HPDSEND to take the "current enhancement"; it is not currently used by HPDSEND. It is recommended that you set this variable to spaces or binary zeros.

OR

labeldescrpt is made up of the following set of subparameters:

labelset A 36-byte character array specified by double backslashes followed by the USASCII character label set name.

Specifying (double backslash)$RESET_FORM tells Hi-Li to reset function key labels to the base set specified for the current form. If a label set was not defined for the form, Hi-Li uses the forms file global set as the base set.

Specifying spaces tells Hi-Li to not update the current function key labels.

	For a discussion of the *labelset* subparameter, see the Discussion for HPDSEND that follows these parameters.
labelbuf	A byte array of characters that identifies for HPDSEND the application area that contains the label strings. For the HP3000, 16 bytes are sent to each label.

Discussion

HPDSEND is one of the three core intrinsics that make up the screen management intrinsics. Its role is to establish the overall context for an interactive transaction. The context is established by getting the form. It determines the form's display position, initializes fields and loads application data, places an application message in the message window, displays the screen and key labels, and places the cursor at a selected input field.

Parameters that are not required by your application can be compacted and passed as abbreviated parameters.

Creating Function Key Label Sets Using LABELSET. Because there is no facility within FORMSPEC/V for defining free- floating function key label sets,the HPDPROMPT and HPDSEND *labelset*,a subparameter of the *labeldescrpt* parameter, provides this ability by "borrowing" the function key labelset from the form you specify in the*labelset* subparameter.

This form can be any of the "live" forms in the forms file or it can be a "dummy" form which exists only as a place holder for the function key label set. You can create a place holder for a key label set by adding a form to the forms file, defining a set of labels for the form, and building a one-character, text-only, screen image for the dummy form.

Examples

COBOL:

```
01   formpak.
     05  formname            pic x (32).
     05  formposition        pic s9(8) comp.
     05  chnglisttype        pic s9(8) comp.
     05  listcount           pic s9(8) comp.
     05  chngentry     occurs 5 times.
         10  fieldident      pic x(32)
         10  chngtype        pic s9(8) comp.
         10  chngspec        pic s9(8).

01   labeldescrpt.
     05  descrptcnt            pic s9(8) comp.
     05  descrptentry  occurs 8 times.
        10  labelident        pic s9(8) comp.
        10  labelenh          pic x(8).

01   labelbuf.
     05 labelentry  occurs 8 times.
        10  labeltext        pic X(16).
```

FORTRAN:

```
INTEGER*4        FORMPAK(44)
CHARACTER*32     FORMNAME
INTEGER*4        FORMPOSITION
```

```
      INTEGER*4        CHNGLISTTYPE
      INTEGER*4        LISTCOUNT
      EQUIVALENCE (FORMPAK(1), FORMNAME),
     +            (FORMPAK(9), FORMPOSITION),
     +            (FORMPAK(10), CHNGLISTTYPE),
     +            (FORMPAK(11), LISTCOUNT)
          CHARACTER*44  FIELDIDENT(1,3)
          EQUIVALENCE (FORMPAK(12), FIELDIDENT)
          INTEGER*4     CHNGTYPE (11,3)
          CHARACTER*4   CHNGSPEC (11,3)
          EQUIVALENCE (FORMPAK(12)  CHNGTYPE),
     +                (FORMPAK(12), CHNGSPEC)

      INTEGER*4 LABELDESCRPT (25)
      INTEGER*4 DESCRPTCNT
      EQUIVALENCE (LABELDESCRPT(1), DESRPTCNT)
         INTEGER*4 LABELIDENT  (3,8)
         CHARACTER*4 LABELENH (3,8)
         EQUIVALENCE (LABELDESCRPT(3), LABELIDENT),
                     (LABELDESCRPT(3), LABELENH)
```

For an example of how a record structure can be manipulated in FORTRAN, see the COLLECT_TXNS subroutine in the FORTRAN example program in Appendix C.

```
      INTEGER*4      LABELBUF(32)
      CHARACTER*16   LABELTEXT(1,8)
      EQUIVALENCE  (LABELBUF(1), LABELTEXT)
```

Pascal:

```
type
  chngentry_rec = record
    fieldident      : packed array [1..32] of char;
    chngtype        : integer;
    chngspec        : packed array [1..8] of char;
      end;

formpak_rec = record
  formname        : packed array [1..32] of char;
  formposition    : integer;
  chnglisttype    : integer;
  listcount       : integer;
  chngentry       : array [ 1..5] of chngentry_rec;
    end;

var
  formpak           : formpak_rec

type
  descrptentry_rec = record
    labelident        : integer;
    labelenh          : packed array [1..8] of char;
      end;

labeldescrpt_rec = record
  descrptcnt        : integer;
  descrptentry      : array [1..8] of
                    descrptentry_rec;
    end;

var
  labeldescrpt        : labeldescrpt_rec;

type
  labelbuf_rec = record
    labeltext     : array [1..8] of
                    packed array [1..16] of char;

var
  labelbuf        : labelbuf_rec;
```

HPDSETENV

Allows your application to configure elements of the current execution environment.

Syntax

HPDSETENV *globalpak, returnpak, mode, envbufdescrpt, envbuf*

Parameter Descriptions

globalpak — A compound parameter that passes information between intrinsics. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

returnpak — A compound parameter that passes information back from the intrinsic to the application. For a complete description of this parameter, see "Global Parameters" in Chapter 3.

mode — A four-byte integer returned to the intrinsic that specifies the element in the execution environment that you want to configure. The values you can specify are:

2 = native language identifier

3 = auto test option

78 = switch device out of block mode

79 = switch device into block mode

envbufdescrpt — A four-byte integer that is input to the intrinsic. Use this parameter for those cases when your application is allowed to specify the size and arrangement of the configuration update passed in the *envbuf* parameter.

Note: The input format of most of the available configuration updates is fixed. If this is the case, this parameter is ignored.

envbuf — A compound parameter that is input to the intrinsic. It defines the area that contains the configuration update. The value you supply for this parameter depends on the *mode* you have specified:

(If) Mode	Element	(Supply) Length/format
2	Native language	A four-byte integer that specifies the language conventions to be used. To use this feature, the forms file must be open and your application must ensure the following: - An "international" forms file is currently being accessed. - Native language subsystem software is installed on the system you are using. - The native language you are requesting must be on the system. See the *Native Language Support Reference Manual* (Part No. ?) for the language codes that you can use.
3	Auto test option	A four-byte integer that sets up automatic terminal

		interaction. To use this mode, the forms file must have been opened or the device enabled. The values you can supply are: 0 = disable 1 = enable
78	Switch out of block mode	An 88-byte character array that is the name of the device file you want to switch out of block mode. To use this mode, the device must have been enabled by HPDENABLETERM.
79	Switch into block mode	An 88-byte character array that is the name of the device file you want to switch into block mode. To use this feature, the device must have been enabled and switched out of block mode.

Discussion

HPDSETENV is designed for advanced programming applications. It allows you to configure various elements of your application's current execution
environment. You can configure four elements in your application's environment. You can:

1. Configure the native language identifier so that prompts and data editing follow the conventions of a particular language.

2. Turn on the auto test feature so that your application can execute without someone having to be at the terminal. One possible use of this feature is for automatic test suites.

3. Momentarily switch a device into

4. - or out of block mode.

HP32424

Returns the version identifier of the installed version of the HP screen management (HP32424) intrinsics.

Syntax

HP32424 *versionvuf*

Parameter Descriptions

versionbuf	A 14-byte array structure of characters that returns the product and version number in the format HP32424v.uu.ff.

Discussion

Use this intrinsic if your application needs to ensure that a particular version of Hi-Li is available.

Chapter 4 Data Mapping Methods

This chapter documents the six ways in which you can map data that you are moving between your application and the fields on a form. You indicate how you want to map the data by the values you assign the *datadescrpt* parameter of the HPDDSEND, HPDREAD, or HPDPRINTFORM intrinsic.

How you choose to map data depends on whether you are moving some, all, or none of the data, whether or not your application requires data type conversions, and on any special run-time requirements your application may have. The six mapping methods can be divided into three groups, which are:

1. Transfer all or none of the data, or transfer a subset of the data. Data methods A, B, and C, are examples of this kind of data transfer. These methods do not perform type conversion: all data is transferred as type CHAR.

2. Transfer data or a subset of data and provide data type conversion within your application. Data methods D and E are examples of this kind of data transfer.

3. Transfer data and perform data type conversions using an Application-ready Buffer (ARB). ARBs are defined in forms files and then compiled: they cannot be changed at run-time. Data method F is an example of this kind of transfer. Each of the six data mapping methods, A through F, are discussed below.

Data Type Conversion

If you plan on using data transfer method F, that uses an ARB as the means for data type conversion, you must create an ARB. It presents data as an application expects to use it. You create an ARB in FORMSPEC/V by supplying information on menus. This eliminates the need for you to code conversion instructions in your application. Refer to the *HP Data Entry and Forms Management System (VPLUS/V) Reference Manual* (Part No. 32209-90001) for complete instructions on creating an ARB.

The DATADESCRPT Parameter

The *datadescrpt* parameter is a compound parameter used in the HPDREAD, HPDSEND, and HPDPRINTFORM intrinsics to tell Hi-Li how you want to transfer data between your application and a form. It is made up of ten

subparameters. The first four subparameters supply header information; the last six provide table information.

You supply values for some or all of these subparameters, depending on the mapping method you are using. In some cases, a subparameter is not used by a particular method, but it must be passed. The particular values that you can supply for a particular mapping method are discussed under that method.

These are the ten *datadescrpt* subparameters:

descrpttype	A four-byte integer that tells Hi-Li the data mapping method you want to use.
buflen	A four-byte integer that indicates the number of bytes of buffer data you want to move.
rtnbuflen	A four-byte integer that indicates the number of bytes of buffer data actually moved.
entrycnt	A four-byte integer that indicates the number of active field entries in the table.
fldentry	Identifies a table you create in your application.
fldid	A four-byte integer or 32-byte character array which identifies a field.
fldloc	A four-byte integer that gives the field location by indicating the offset of the field within the buffer.
fldlen	A four-byte integer that indicates the number of bytes of field data you want to move.
rtnfldlen	A four-byte integer that indicates the number of bytes of field data actually moved.
typcnvcode	A four-byte integer that tells Hi-Li the type of data conversion you want performed.

Data Transfer Method A

Data transfer method A is the simplest way of indicating data transfer. You can transfer all data or no data. You indicate a transfer of all data by providing the value 0 for *datadescrpttype* or a transfer of no data with a value of -1. When all data is moved, it is moved as a concatenated string to or from the *databuf* parameter. The number of bytes moved is equal to the sum of the lengths of all the fields on the form. No data type conversion takes place; all data is transferred as type CHAR.

To indicate this method of transfer, you must supply the *descrpttype* subparameter and one of the following values:

-1 = transfer no data
 0 = transfer all data

Examples

```
‘screen 10‘ COBOL: 01 datadescrpt pic s9(8) comp.

FORTRAN: INTEGER*4 DATADESCRPT

Pascal:  var datadescrpt :  integer;
```

Data Transfer Method B

This method of data transfer lets you move a subset of data starting with the first byte. You indicate this method by supplying 10 as the four-byte integer for *descrpttype*. Data is moved as a concatenated string from or to *databuf*. This method allows for no data type conversions.

The subparameters you must supply to use this data mapping method and their range of values are:

descrpttype 10

buflen Supply the number of bytes you want to move.

rtnbuflen The number of bytes actually moved is returned in this subparameter.

Examples

```
     COBOL:
                  01 datadescrpt.
                     05  descrpttype       pic s9(8) comp.
                     05  buflen            pic s9(8) comp.
                     05  rtnbuflen         pic s9(8) comp.

         FORTRAN:
                  INTEGER*4     DATADESCRPT(3)
                  INTEGER*4     DESCRPTTYPE
                  INTEGER*4     BUFLEN
                  INTEGER*4     RTNBUFLEN
                  EQUIVALENCE (DATADESCRPT(1), DESCRPTTYPE),
                 +            (DATADESCRPT(2), BUFLEN),
                 +            (DATADESCRPT(3), RTNBUFLEN)

        Pascal:
                  type
                    datadescrpt_rec = record
                      descrpttype        : integer;
                      buflen             : integer;
                      rtnbuflen          : integer;
                        end;

                  var
      /              datadescrpt          : datadescrpt_rec;
```

Data Transfer Method C

This data transfer method lets you move a subset of data from or to the fields referenced in the form. Use this method if your application initializes information from both the forms file and the application buffer, or if you want to return a selected subset of data to your application. You indicate this method by specifying the value 20 or 30 for the *descrpttype* subparameter. Use 20 if you want to identify fields

by number and 30 if you want to identify fields by name.

The offset and length of each field in the application data buffer is implicitly defined by the form. This means the application buffer must have the same layout as the form.

The subparameters you must supply to use this data mapping method and their values are:

descrpttype — Supply a value of 20 to identify fields by number.

Supply a value of 30 to identify fields by name.

buflen — A four-byte integer. This subparameter is not used by Hi-Li for this method, but must be passed anyway.

rtnbuflen — A four-byte integer. This subparameter is not used by Hi-Li for this method, but must be passed anyway.

entrycnt — Supply a four-byte integer to represent the number of active field entries in the table.

fldentry — Identifies the table in your application.

fldid — If you supplied the value 20 for *descrpttype*, supply a four-byte integer which identifies the field. If the integer is greater than zero, it indicates the field number. If the integer is less than zero, it indicates screen order.

If you supplied the value 30 for *descrpttype*, supply a 32-byte character array that is the USASCII name of the field.

Examples

COBOL:

```
01 datadescrpt.
   05  descsrpttype      pic s9(8) comp.
   05  buflen            pic s9(8) comp.
   05  rtnbuflen         pic s9(8) comp.
   05  entrycnt          pic s9(8) comp.
   05  fldentry   occurs 5 times.
      10  fldid         pic x(32).
```

FORTRAN:

```
INTEGER*4     DATADESCRPT(44)
INTEGER*4     DESCRPTTYPE
INTEGER*4     BUFLEN
INTEGER*4     RTNBUFLEN
INTEGER*4     ENTRYCNT
EQUIVALENCE (DATADESCRPT(1), DESCRPTTYPE),
+           (DATADESCRPT(2), BUFLEN),
+           (DATADESCRPT(3), RTNBUFLEN),
+           (DATADESCRPT(4), ENTRYCNT)
      CHARACTER*32  FLDID(1,5)
      EQUIVALENCE (DATADESCRPT(5), FLDID)
```

For an example of how a record structure can be manipulated in FORTRAN, see the COLLECT_TXNS subroutine in the FORTRAN example program in Appendix C.

```
Pascal:     type
              fldentry_rec = record
                fldid           : packed array [1..32] of char;
                  end;

            datadescrpt_rec = record
              descrpttype     : integer;
              buflen          : integer;
              rtnbuflen       : integer;
              entrycnt        : integer;
              fldentry        : array [1..5] of fldentry_rec;
                end;

            var
              datadescrpt         : datadescrpt_rec;
```

Data Transfer Method D

This data transfer method lets you move data to and from the fields referenced in the form, but the application buffer does not have to match the screen layout. All data must come from one buffer. This method also allows your application to specify data type conversions. It has all the power and and usage of method F (that uses an ARB), but has the advantage of letting your application resolve data buffer formats and layout at run-time. Use this transfer method when the application data buffer does
not correspond to the form buffer and your application has special run-time requirements.

You indicate this kind of data transfer by assigning the value 40 or 50 to the subparameter *descrpttype*. Assign the value 40 if you want to identify fields by number. Assign the value 50 if you want to identify fields by name.

The subparameters you must supply to use this data mapping method and their values are:

descrpttype	Supply the value 40 to identify fields by number. Supply the value 50 to identify fields by name.
buflen	A four-byte integer. This subparameter is not used by Hi-Li for this method, but must be passed anyway.
rtnbuflen	A four-byte integer. This subparameter is not used by Hi-Li for this method, but must be passed anyway.
entrycnt	Supply a four-byte integer to indicate the number of active field entries in the table.
fldentry	Identifies the table in your application.
fldloc	Supply a four-byte integer to indicate the byte offset of the field with respect to the *databuf* parameter. The offset is 1 relative. If you supply a zero, the offset of the referenced field in the form is used by default. You cannot supply a negative offset.
fldlen	Supply a four-byte integer to indicate the

number of bytes you want to move. If you supply a zero, the length of the referenced field in the form data buffer is used by default. If you supply a negative value, no data is moved between the application and the form.

rtnfldlen A four-byte integer. The number of field data bytes actually moved is returned here.

typcnvcode Supply a four-byte integer to indicate the data type conversion you want performed. The data type conversion codes are:

0 = no conversion
1 = two-byte integer conversion
2 = four-byte integer conversion
3 = four-byte HP3000 format floating point (real) conversion
4 = eight-byte HP3000 format floating point (long) conversion
5 = reserved for four-byte IEEE format floating point (real) conversion
6 = reserved for eight-byte IEEE format floating point (long) conversion

fldid If you have given *descrpttype* a value of 40, supply a four-byte integer that identifies a field. An integer greater than zero indicates a field number; an integer less than zero indicates screen order.

If you have given *descrpttype* a value of 50, supply a 32-byte character array that represents the USASCII name of the field.

Examples

COBOL:

```
 01  datadescrpt.
     05  descrptype         pic s9(8) comp.
     05  buflen             pic s9(8) comp.
     05  rtnbuflen          pic s9(8) comp.
     05  entrycnt           pic s9(8) comp.
     05  fldentry       occurs 5 times.
         10  fldloc         pic s9(8) comp.
         10  fldlen         pic s9(8) comp.
         10  rtnfldlen      pic s9(8) comp.
         10  typcnvcode     pic s9(8) comp.
         10  fldid          pic x(32).
```

FORTRAN:

```
      INTEGER*4     DATADESCRPT(64)
      INTEGER*4     DESCRPTTYPE
      INTEGER*4     BUFLEN
      INTEGER*4     RTNBUFLEN
      INTEGER*4     ENTRYCNT
      EQUIVALENCE (DATADESCRPT(1), DESCRPTTYPE(,
     +            (DATADESCRPT(2), BUFLEN),
     +            (DATADESCRPT(3), RTNBUFLEN),
     +            (DATADESCRPT(4), ENTRYCNT)
       INTEGER*4     FLDLOC(12,5)
       INTEGER*4     FLDLEN(12,5)
       INTEGER*4     RTNFLDLEN(12,5)
       INTEGER*4     TYPCNVCODE(12,5)
       EQUIVALENCE (DATADESCRPT(5), FLDLOC),
      +            (DATADESCRPT(5), FLDLEN),
      +            (DATADESCRPT(5), RTNFLDLEN),
      +            (DATADESCRPT(5), TYPCNVCODE)
```

```
                    CHARACTER*16  FLDID(3,5)
        EQUIVALENCE (DATADESCRPT(5), FLDID)
```

For an example of how a record structure can be manipulated in FORTRAN, see the COLLECT_TXNS subroutine in the FORTRAN example program in Appendix C.

Pascal:

```
type
  fldentry_rec = record
    fldloc          : integer;
    fldlen          : integer;
    rtnfldlen       : integer;
    typcnvcode      : integer;
    fldid           : packed array [1..32] of char;
      end;

datadescrpt_rec = record
  descrpttype     : integer;
  buflen          : integer;
  rtnbuflen       : integer;
  entrycnt        : integer;
  fldentry        : array [1..5] of fldentry_rec;
    end;

var
  datadescrpt       : datadescrpt_rec;
```

Data Transfer Method E

This data transfer method moves data to and from fields referenced in the form and allows your application to specify data type conversions. This method differs from Method D in that transferred data can come from more than one buffer. Use method E when you need to transfer data between a form and multiple buffers in the application.

The application passes the field address contained in the application data space, the field length, and the data conversion operation that is to be performed. For the HP 3000 MPE/V operating system, the address you pass must be within plus or minus 32000 of the *databuf* address, and must map to a location within the same data segment as *databuf*. The address must also be DB-relative.

You indicate this kind of data transfer by assigning the value 60 or 70 to the subparameter *descrpttype*. Assign the value 60 if you want to identify fields by number. Assign the value 70 if you want to identify fields by name.

The values you must supply to use this data mapping method are:

descrpttype	Supply the value 60 to identify fields by number. Supply the value 70 to identify fields by name.
buflen	A four-byte integer. This subparameter is not used by Hi-Li for this method, but must be passed anyway.
rtnbuflen	A four-byte integer. This subparameter is not used by Hi-Li for this method, but must be

passed anyway.

entrycnt Supply a four-byte integer that represents the number of active field entries in the table.

fldentry Identifies the table defined by your application.

fldloc Supply a four-byte integer that represents the byte address of the field in the application data space. If you supply a zero, Hi-Li assumes the field is in *databuf* and the location of the field in the application data space is determined by applying the offset of the referenced field in the form data buffer to *databuf*.

fldlen Supply a four-byte integer to represent the number of bytes you want to move. If you supply a zero, the length of the referenced field in the form data buffer is used by default. If you supply a negative 1, no data is moved between the application and the form.

rtnfldlen A four-byte integer. The number of field data bytes actually moved is returned here.

typcnvcode Supply a four-byte integer to indicate the data type conversion you want performed. The conversions that you can use are:

0 = no conversion
1 = two-byte integer conversion
2 = four-byte integer conversion
3 = four-byte HP3000 format floating point (real) conversion
4 = eight-byte HP3000 format floating point (long) conversion
5 = reserved for four-byte IEEE format floating point (real) conversion
6= reserved for eight-byte IEEE format floating point (long) conversion

fldid If you have given *descrpttype* a value of 60, supply a four-byte integer that identifies a field. An integer greater than zero indicates a field number; an integer less than zero indicates screen order.

If you have given *descrpttype* a value of 70, supply a 32-byte character array that represents the USASCII name of the field.

Examples

COBOL:

```
01  datadescrpt.
    05  descrptype         pic s9(8) comp.
    05  buflen             pic s9(8) comp.
    05  rtnbuflen          pic s9(8) comp.
    05  entrycnt           pic s9(8) comp.
    05  fldentry       occurs 5 times.
        10  fldloc         pic s9(8) comp.
        10  fldlen         pic s9(8) comp.
        10  rtnfldlen      pic s9(8) comp.
        10  typcnvcode     pic s9(8) comp.
        10  fldid          pic x(32).
```

FORTRAN:

```
INTEGER*4     DATADESCRPT(64)
INTEGER*4     DESCRPTTYPE
INTEGER*4     BUFLEN
```

```
      INTEGER*4     RTNBUFLEN
      INTEGER*4     ENTRYCNT
      EQUIVALENCE (DATADESCRPT(1), DESCRPTTYPE(,
     +            (DATADESCRPT(2), BUFLEN),
     +            (DATADESCRPT(3), RTNBUFLEN),
     +            (DATADESCRPT(4), ENTRYCNT)
            INTEGER*4     FLDLOC(12,5)
            INTEGER*4     FLDLEN(12,5)
            INTEGER*4     RTNFLDLEN(12,5)
            INTEGER*4     TYPCNVCODE(12,5)
            EQUIVALENCE (DATADESCRPT(5), FLDLOC),
           +            (DATADESCRPT(5), FLDLEN),
           +            (DATADESCRPT(5), RTNFLDLEN),
           +            (DATADESCRPT(5), TYPCNVCODE)
            CHARACTER*16  FLDID(3,5)
            EQUIVALENCE (DATADESCRPT(5), FLDID)
```

For an example of how a record structure can be manipulated in FORTRAN, see the COLLECT_TXNS subroutine in the FORTRAN example program in Appendix C.

Pascal:

```
type
  fldentry_rec = record
    fldloc          : integer;
    fldlen          : integer;
    rtnfldlen       : integer;
    typcnvcode      : integer;
    fldid           : packed array [1..32] of char;
      end;

datadescrpt_rec = record
  descrpttype     : integer;
  buflen          : integer;
  rtnbuflen       : integer;
  entrycnt        : integer;
  fldentry        : array [1..5] of fldentry_rec;
    end;

var
  datadescrpt       : datadescrpt_rec;
```

Data Transfer Method F

This method lets you transfer data and perform data type conversions using an ARB you have defined in the forms file. Use method F if your application requires data type conversions and only one buffer is used to
transfer data.

You indictate this data transfer method by assigning the value 1000 or 1100 to *descrpttype*. The value 1000 tells Hi-Li to convert real (floating point) numbers according to HP 3000 format rules. The value 1100 tells Hi-Li to convert real (floating point) numbers according to IEEE format rules.

The values you must supply to use method F are:

descrpttype Supply a value of 1000 to convert real numbers using MPE formatting rules.

Supply a value of 1100 to convert real numbers

using IEEE formatting rules.

Examples

```
COBOL:      01   datadescrpt          pic s9(8) comp.

FORTRAN:    INTEGER*4     DATADESCRPT

Pascal:     var
     datadescrpt          : integer;
```

Summary of Data Mapping Methods

Methods A, B, and C allow you to transfer data and subsets of data, but do not allow data type conversion. Method A is the simplest of the six methods: you can choose to transfer all data or no data. Methods B and C allow you to transfer subsets of data. Method B transfers bytes sequentially, starting at the beginning of the buffer. Method C allows you to select specific fields of data from the buffer, not necessarily from the beginning of the buffer or in sequence.

Methods D, E, and F all allow for data type conversion. Methods D and F are alike except that D requires that the conversion instruction be coded into your application. The advantage of method D is that it allows for the run-time resolution of buffer formats and layout. Method F uses an Application-ready Buffer to perform data type conversions. Unless your application has special run-time requirements, it is recommended that you use method F. Method E requires that conversion instructions be coded into your application. It is more powerful than either D or F because it allows you to transfer data from more than one buffer. Table 4-1 summarizes the six methods.

Table 4-1. Data Mapping Summary

FEATURE	A 0-1	B 10	C 20-30	D 40-50	E 60-70	F 1000-1100
Receive or send full data buffer	X	X				
Buffer length specified by application		X				
Actual buffer length returned to application		X				
Receive or send no data	X					
Receive or send data by field			X X	X X	X X	
Application may specifies field length				X X	X X	

FEATURE	0-1	A 10	B 20-30	C 40-50	D 60-70	E 1000-1100	F
Actual field length returned by application				X X	X X		
Receive or send no data by field				X X	X X		
Field identified by name			X	X	X		
Field identifed by number or screen order			X	X	X		
Application may specify data type conversion				X X	X X		
Can mix field data initialization from forms and the application				X X	X X	X	X
Supports data transfer between a form and multiple application buffers					X X		
Data transfers and conversions between a form and the application buffer using forms file rules						X	X
Conversion of real numbers according to IEEE format rules				X X	X X		X
Conversion of real numbers according to HP 3000 format rules				X X	X X	X	

Appendix A Error Messages

Hi-Li Errors and Exceptions

See the *returnpak* parameter under "Common Parameters" for a description of error and exception reporting mechanisms.

There are a number of cases where the error or exception code returned by the Hi-Li intrinsics is an "um brella" for errors returned by the underlying facilities. These cases are indicated by the commercial 'at' sign (@) to the right of the error or exception number. In these cases, the message text returned in the ‘i‘returnmsg‘n‘ argument of the *returnpak* parameter is generated by the underlying facility, and is more
specific than the corresponding text below. A cross-reference for these cases follows the list below.

HPDSEND Errors and Exceptions:

-46 Unrecognized data type conversion code passed.

-45 Unrecognized data description type code passed.

-43 Buffer length passed does not match form data buffer length.

-38 Unrecognized window enhancement code passed.

-35 @ Attempt to change field characteristics failed.

-31 @ Unable to reset form function key label set.

-30 @ Unable to retrieve named function key label set.

-28 @ Function key labeling failed.

-27 Terminal was switched out of block mode (and is not "enabled").

-25 @ Field highlighting failed.

-24 Unable to do transformations according to forms file rules.

-21 @ Field data initialization failed.
-20 @ Get of form failed.

-19 @ Unrecognized form positioning code passed.

-18 Unable to decode the form name that was passed.

-17 Terminal is not enabled.

-16 @ Copying of application data to form failed.

-15 @ Loading of application message into window area failed.

-14 @ Screen painting failed.

-13 @ Cursor positioning failed.

-12 @ Attempt to obtain information about field failed.

-9 Forms file is not open.

-6 Communications area is not setup.

30 @ Field initialization data errors detected.

HPDREAD Errors and Exceptions:

-46 Unrecognized data type conversion code passed.

-45 Unrecognized data description type code passed.

-40 @ Attempt to obtain field error bitmap failed (internal error).

-37 Read call out of sequence.

-34 @ Field data reformatting (finishing) failed.

-32 @ Field data editing failed.

-27 Terminal was switched out of block mode (and is not "enabled").

-24 Unable to do transformations according to forms file rules.

-23 @ Read failed.

-22 @ Copying of data from form to application area failed.

-17 Terminal is not enabled.

-12 @ Attempt to obtain information about field failed.

-9 Forms file is not open.

-6 Communications area is not setup.

24 Read timed out.

31 Inactive function key (or field) terminated read.

33 @ Field editing data errors detected.

35 @ Field reformatting (finishing) data errors detected.

37 @ Buffer transformation warning detected.

HPDPROMPT Errors and Exceptions:

-40 @ Attempt to obtain field error bitmap failed (internal error).

-39 Prompt call may only follow Send, Read, or another Prompt call.

-38 Unrecognized window enhancement code passed.

-31 @ Unable to reset form function key label set.

-30 @ Unable to retrieve named function key label set.

-28 @ Function key labeling failed.

-27 Terminal was switched out of block mode (and is not "enabled").

-25 @ Field highlighting failed.

-22 @ Attempt to reset field in error failed (internal error).

-17 Terminal is not enabled.

-16 @ Attempt to reset field in error failed (internal error).

-15 @ Loading of application message into window area failed.

-14 @ Screen painting failed.

-13 @ Cursor positioning failed.

-12 @ Attempt to obtain information about field failed.

-9 Forms file is not open.

-6 Communications area is not setup.

HPDOPENFORMS Errors and Exceptions:

-10 Attempt to open forms file which is currently open.

-5 Inappropriate file name passed.

-3 @ Forms file open failed.

-2 Access denied; contact the person who supports use of this software.

-1 Unrecognized call protocol (programming language code) passed.

HPDCLOSEFORMS Errors and Exceptions:

-9 Forms file is not open.

-7 @ Forms file close failed.

-6 Communications area is not setup.

-5 Inappropriate file name passed.

HPDENABLETERM Errors and Exceptions:

-11 Attempt to enable terminal which is currently enabled.

-5 Inappropriate file name passed.
-4 @ Terminal enable failed.

-2 Access denied; contact the person who supports use of this software.

-1 Unrecognized call protocol (programming language code) passed.

5 Your terminal is not supported.

HPDDISABLETERM Errors and Exceptions:

-27 Terminal was switched out of block mode (and is not "enabled").

-17 Terminal is not enabled.

-8 @ Terminal disable failed.

-6 Communications area is not setup.

-5 Inappropriate file name passed.

HPDGETDESIGN Errors and Exceptions:

-3xx Invalid output buffer length parameter passed.

Thus: -301 = invalid length for file output;
-302 = invalid length for form output;
-303 = invalid length for field output;
etc.

-2xx Unrecognized key descriptor passed.

Thus: -201 = unrecognized file level key descriptor;
-202 = unrecognized form level key descriptor;
-203 = unrecognized field level key descriptor;
etc.

-1xx No retrieval key specified.

Thus: -101 = no file key;
-102 = no form key;
-103 = no field key;
etc.

-55 Invalid use of private mode.

-54 @ Attempt to obtain field level design information failed.

-53 @ Attempt to obtain form level design information failed.

-52 @ Attempt to obtain file level design information failed.

-51 May not pass GLOBALPAK from other intrinsics to this intrinsic.

-50 Unrecognized mode parameter passed.

-22 @ Copying of data from form to application area failed.

-21 @ Field data initialization failed.

-20 @ Get of form failed.

59 @ Field contents flagged in error by initialization process.

HPDGETENV and HPDSETENV Errors and Exceptions:

-68 Unrecognized auto test option code passed.

-65 Terminal has not been switched out of block mode.

-64 @ Attempt to switch device into block mode failed.

-63 @ Attempt to switch device out of block mode failed.

-62 Language characteristic may not be set for current forms file.

-60 @ Attempt to configure language failed.

-58 @ Attempt to return forms file language characteristic failed.

-56 Unrecognized configuration mode code passed.

-27 Terminal was switched out of block mode (and is not "enabled").

-17 Terminal is not enabled.

-9 Forms file is not open.

-6 Communications area is not setup.

-5 Inappropriate file name passed.

HPDPRINTFORM Errors and Exceptions:

-88 Attempt to advance list file page failed.

-86 Attempt to open list file failed.

-84 unuseable field identifier passed.

-82 Load of field data failed.

-80 @ Unload of field data failed.

-78 Unrecognized format code for field identifier passed.

-76 Unrecognized fill description code passed.

-74 Unrecognized reformat code passed.

-72 Unrecognized page control code passed.

-70 Unrecognized underline control code passed.

-45 Unrecognized data description type code passed.

-34 @ Field data reformatting (finishing) failed.

-21 @ Field data initialization failed.

-20 @ Get of form failed.

-18 Unable to decode the form name that was passed.

-17 Terminal is not enabled.

```
-16 @  Copying of application data to form failed.
-12 @  Attempt to obtain information about field failed.

 -9    Forms file is not open.

 -6    Communications area is not setup.

 30 @  Field initialization data errors detected.

 35 @  Field reformatting (finishing) data errors detected.

 40    May not have filled all character positions in field(s).
```

Errors and Exceptions Cross-Reference

HP3000

Underlying facility by Hi-Li version:

1. A.@@.@@ - VPLUS/V

For indicated underlying facility, generating procedure by Hi-Li return code (status returned by generating procedure is in 'sublayerstatus' argument of 'RETURNPAK' parameter):

```
VPLUS/V:     -82  Vputfield
             -80  Vgetfield
             -64  Vopenterm
             -63  Vcloseterm
             -60  Vsetlang
             -58  Vgetlang
             -54  Vgetfieldinfo
             -53  Vgetforminfo
             -52  Vgetfileinfo
             -40  Vgetforminfo
             -35  Vchangefield
             -34  Vfinishform
             -32  Vfieldedits
             -31  Vgetkeylabels
             -30  Vgetkeylabels
             -28  Vsetkeylabel
             -25  Vseterror
             -23  Vreadfields
             -22  Vgetfield, Vgetbuffer, or Vget types
             -21  Vinitform
             -20  Vgetnextform
             -19  Vprintform
             -16  Vputfield, Vputbuffer, or Vput types
             -15  Vputwindow
             -14  Vshowform
             -13  Vplacecursor
```

```
  -12  Vgetfieldinfo
  -8   Vcloseterm
  -7   Vcloseformf
  -4   Vopenterm
  -3   Vopenformf
  +30  Vinitform   (data initialization error)*
  +33  Vfieldedits (data editing error)*
  +35  Vfinishform (data finishing error)*
 +37  Vgetbuffer  (buffer transformation warning)
  +59  Vgetfield   (field data content flagged)
```

* 'sublayerstatus' will be zero; 'returnmsg' argument of 'RETURNPAK' parameter will contain designer's custom message (or standard catalog message if no custom message defined).

Appendix B VPLUS/V and HI-LI Intrinsics

Comparison of VPLUS/V and Hi-Li Intrinsics

The Hi-Li intrinsics consist of eleven intrinsics that provide a similar functionality to the current VPLUS/V intrinsics. Table B-1 lists each Hi-Li intrinsic and the corresponding VPLUS/V intrinsics that perform the same function. This table will be helpful if you currently use VPLUS/V intrinsics and plan to convert to using Hi-Li intrinsics.

Table B-1. Comparison of VPLUS/V and Hi-Li Intrinsics

Hi-Li Intrinsic	VPLUS/V Intrinsics
HPDSEND	VCHANGEFIELD VGETNEXTFORM VINITFORM VPLACECURSOR VPUTBUFFER VPUTFIELD VPUTTYPE VPUTWINDOW VSETKEYLABEL VSHOWFORM
HPDREAD	VFIELDEDITS VFINISHFORM VGETBUFFER VGETFIELD VGETTYPE VREADFIELDS
HPDPROMPT	VPLACECURSOR VPUTWINDOW VSETERROR VSETKEYLABEL VSHOWFORM
HPDOPENFORMS	VOPENFORMF
HPDCLOSEFORMS	VCLOSEFORM
HPDENABLETERM	VOPENTERM
HPDDISENABLETERM	VCLOSETERM

HPDPRINTFORM	VFINISHFORM VGETNEXTFORM VINITFORM VPRINTFORM VPUTBUFFER VPUTFIELD VPUTTYPE

Comparison of VPLUS/V and Hi-Li Intrinsics (cont.)

Hi-Li Intrinsic	**VPLUS/V Intrinsics**
HPDGETENV	VGETLANG
HPDSETENV	VSETLANG VTURNOFF VTURNON
HPDGETDESIGN	VCLOSEFORMF VGETFIELD VGETFIELDINFO VGETFILEINFO VGETFORMINFO VINITFORM VOPENFORMF

Appendix C Diagnostics and Sample Programs

This appendix discusses the Hi-Li diagnostic facility and version utility, and provides sample application programs written in COBOL, FORTRAN, and Pascal that demonstrate the use of Hi-Li intrinsics.

Hi-Li Diagnostics Trace Facility

The Hi-Li intrinsic diagnostic facility is an application debugging tool that can be used to diagnose applications that use Hi-Li intrinsics. It shows:

* intrinsic entry and exit,
* intrinsic formats, and
* parameter contents.

To enable the Hi-Li trace facility, set the DEXLIBTRACE job control word to 1.

```
SETJCW DEXLIBTRACE,1 Return
```

Diagnostic information is written to the file HPDTRACE. Trace messages are appended to the file. Note that subsequent traces are appended to the same file. You may want to purge the trace file (HPDTRACE) before performing subsequent traces.

To save a trace file, use the MPE BUILD command to build an 80-character ASCII file. Then, use any editor to view the file. For example,

```
BUILD HPDTRACE; rec = -80,1,F,ASCII
```

You can also direct the trace file to a terminal by using the MPE FILE command. If you do this, the application I/O must be directed to a different terminal. Use this command to redirect the I/O:

```
FILE HPDTRACE = $STDLIST
```

The following is a sample trace for an application using Hi-Li intrinsics.

Example of a Trace File

```
>>>> Entered  HPDOPENFORMS
 Globalpak.expectedvuf  = A.00.10
          .callprotocol =           0
          .comarealen   =         300
```

```
  Formsfile = PAYROLL.WORK.ADMIN
<<<< Leaving HPDOPENFORMS
  Returnpak.returnstatus   =           -3
           .sublayerstatus =           50
>>>> Entered  HPDCLOSEFORMS
  Globalpak.expectedvuf  = A.00.10
           .callprotocol =            0
           .comarealen   =          300
  Formsfile = PAYROLL.WORK.ADMIN
<<<< Leaving HPDCLOSEFORMS
  Returnpak.returnstatus   =           -9
           .sublayerstatus =            0
>>>> Entered  HPDDISABLETERM
  Globalpak.expectedvuf  = A.00.10
           .callprotocol =            0
           .comarealen   =          300
  Termpak.termfile      =
         .bypassfeature =             0
  Devconfig.allolen  =           0
<<<< Leaving HPDDISABLETERM
  Returnpak.returnstatus   =          -17
           .sublayerstatus =            0
>>>> Entered  HPDOPENFORMS
  Globalpak.expectedvuf  = A.00.10
           .callprotocol =            0
           .comarealen   =          300
  Formsfile = PAYROLL.WORK.ADMIN
<<<< Leaving HPDOPENFORMS
  Returnpak.returnstatus   =            0
           .sublayerstatus =            0
>>>> Entered  HPDENABLETERM
  Globalpak.expectedvuf  = A.00.10
           .callprotocol =            0
           .comarealen   =          300
  Termpak.termfile      = HPTERM
         .bypassfeature =             0
  Devconfig.allolen =           0
<<<< Leaving HPDENABLETERM
  Returnpak.returnstatus   =            0
           .sublayerstatus =            0

>>>> Entered  HPDSEND
  Globalpak.expectedvuf  = A.00.10
           .callprotocol =            0
           .comarealen   =          300
  Sendpak.dontenableinput =             0
         .windowenh       =
         .bypassfeature   =             0
  Formpak.formname = DEDUCTION
         .formposition =            0
         .chnglisttype =            2
  Cursorposition (1st 4 bytes as char)    =
        "        (1st 4 bytes as integer) =            0
  Msg.msglen  =           79
     .msgline = Fill in Deduction Transaction according to worksheet.
  Datadescrpt.descrpttype =           -1
  Labeldescrpt (1st 4 bytes as char)  =
  Labeldescrpt (1st 4 bytes as integer)  =           0
  Labelbuf (1st 4 bytes as char) =
<<<< Leaving HPDSEND
  Returnpak.returnstatus   =            0
           .sublayerstatus =            0
>>>> Entered  HPDREAD
  Globalpak.expectedvuf  = A.00.10
           .callprotocol =            0
           .comarealen   =          300
  Readpak.readtime       =            0
         .enablereformat =            1
```

```
         .doread         =            0
         .bypassfeature  =            0
  Readitems.itemcnt        =           0
  Datadescrpt.descrpttype =           10
  Fieldlist.listtype =           0
<<<< Leaving HPDREAD
  Returnpak.returnstatus   =            0
           .sublayerstatus =            0
           .lastitemtype   =            0
           .lastitemnum    =            0
          .lastitemname  =  $ENTER
           .numdataerrs    =            0
  Databuf (1st 4 bytes as char)    = Bert
     "    (1st 4 bytes as integer) =   1113944692
>>>> Entered  HPDSEND
  Globalpak.expectedvuf  = A.00.10
           .callprotocol =            0
           .comarealen   =          300
  Sendpak.dontenableinput =            0
         .windowenh       =
         .bypassfeature   =            0
  Formpak.formname = DEDUCTION
         .formposition =            0
         .chnglisttype =            2

  Cursorposition (1st 4 bytes as char)    =
       "        (1st 4 bytes as integer) =            0
  Msg.msglen  =           79
     .msgline = Fill in Deduction Transaction according to worksheet.
  Datadescrpt.descrpttype =           -1
  Labeldescrpt (1st 4 bytes as char)  =
  Labeldescrpt (1st 4 bytes as integer)  =           0
  Labelbuf (1st 4 bytes as char) =
<<<< Leaving HPDSEND
  Returnpak.returnstatus   =            0
           .sublayerstatus =            0
>>>> Entered HPDREAD
  Globalpak.expectedvuf  = A.00.10
           .callprotocol =            0
           .comarealen   =          300
  Readpak.readtime       =            0
         .enablereformat =            1
         .doread         =            0
         .bypassfeature  =            0
  Readitems.itemcnt        =           0
  Datadescrpt.descrpttype =           10
  Fieldlist.listtype =           0
<<<< Leaving HPDREAD
  Returnpak.returnstatus   =            0
           .sublayerstatus =            0
           .lastitemtype   =            0
           .lastitemnum    =            8
          .lastitemname  =  $PFK_8
           .numdataerrs    =            0
  Databuf (1st 4 bytes as char)    = Bert
     "    (1st 4 bytes as integer) =   1113944692
>>>> Entered  HPDCLOSEFORMS
  Globalpak.expectedvuf  = A.00.10
           .callprotocol =            0
           .comarealen   =          300
  Formsfile = PAYROLL.WORK.ADMIN
<<<< Leaving HPDCLOSEFORMS
  Returnpak.returnstatus   =            0
           .sublayerstatus =            0
>>>> Entered  HPDDISABLETERM
  Globalpak.expectedvuf  = A.00.10
           .callprotocol =            0
           .comarealen   =          300
```

```
  Termpak.termfile      = HPTERM
         .bypassfeature =           0
  Devconfig.allolen  =          0
<<<< Leaving HPDDISABLETERM
  Returnpak.returnstatus   =           0
           .sublayerstatus =           0
```

Version Utility

Hi-Li provides a utility program that you can run if you need to determine what version of the Hi-Li software you are running. To see what version of Hi-Li is installed on your MPE/V system, type the following command following the MPE prompt and press [Return]:

```
:RUN HP32424A
```

The following information is displayed:

```
HP32424 INTRINSICS VERSION:  HP32424A.00.04

        -------------------------
        SEGMENT     VERSION
        -------------------------
          01      A.00.04.S01.00
          02      A.00.04.S02.00
          03      A.00.04.S03.00
          04      A.00.04.S04.00
        -------------------------

END OF PROGRAM
:
```

Sample Programs

The following is a sample application program that illustrates how the various Hi-Li intrinsics are used to develop an interactive application. The same program is developed for three of the languages supported by Hi-Li: COBOL, FORTRAN, and Pascal.

This sample application collects transactions concerning deductions from
an employee payroll and places the edited transactions into a file. Each transaction entered by the operator is subject to the data edits embedded within the input form. The application continues to collect transactions until the operator exits the program or a system error is detected. Keys are defined as follows:

```
Enter key = edit and file transaction
 F1 = exit application
All other  f keys = redo transaction
```

The application has the following structure:

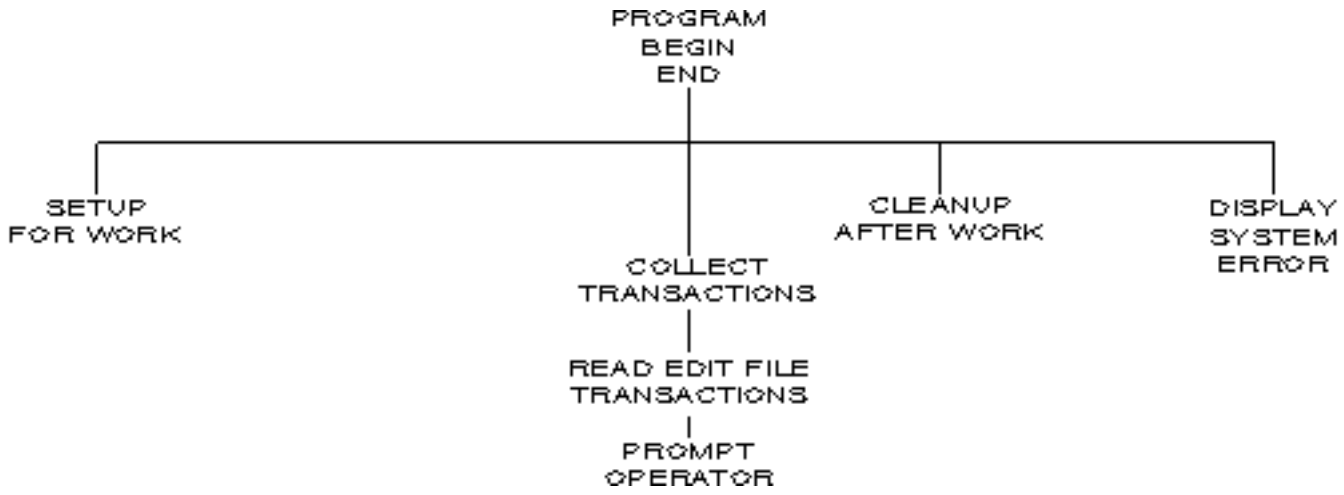

Figure C-1. Structure of the Application Program

COBOL Sample Program

```
      $CONTROL LIST, MAP, VERBS

      IDENTIFICATION DIVISION.

      PROGRAM-ID.  COBOL-EXAMPLE.

     *****
     ***** This application collects employee payroll deduction
     ***** transactions and places the edited transactions into
     ***** a file.
     *****
     ***** For this application:        Enter key = edit and file
     *****                             transaction;
     *****
     *****                                    f8 = exit application;
     *****
     *****                        all other f keys = redo transaction.
     *****
     ***** Each transaction entered by the operator is subjected to
     ***** the data edits embedded within the input form.
     *****
     ***** The application continues to collect transactions until
     ***** either the operator signals to exit or a system error
     ***** is detected.

     *****

      ENVIRONMENT DIVISION.

      INPUT-OUTPUT SECTION.

      FILE-CONTROL.

      SELECT TXN-ENTRY ASSIGN TO "PAYTXN".

      DATA DIVISION.

      FILE SECTION.

      FD  TXN-ENTRY
```

```
    RECORD CONTAINS 200 CHARACTERS
    DATA RECORDS ARE TXN-REC.

01  TXN-REC.
    05  FILLER             PIC X(200).

WORKING-STORAGE SECTION.

01  GLOBALPAK.
    05  EXPECTEDVUF           PIC X(8).
    05  CALLPROTOCOL          PIC S9(8) COMP.
    05  COMAREALEN            PIC S9(8) COMP.
    05  COMAREA               PIC X(300) VALUE LOW-VALUES.

01  RETURNPAK.
    05  RETURNSTATUS          PIC S9(8) COMP.
    05  SUBLAYERSTATUS        PIC S9(8) COMP.
    05  RETURNMSGLEN          PIC S9(8) COMP.
    05  RETURNMSG             PIC X(256).
    05  LASTITEMTYPE          PIC S9(8) COMP.
    05  LASTITEMNUM           PIC S9(8) COMP.
    05  LASTITEMNAME          PIC X(32).
    05  NUMDATAERRS           PIC S9(8) COMP.
    05  NUMCHNGFLDS           PIC S9(8) COMP.

01  FORMSFILE.
    05  FFNAME                PIC X(88).

01  TERMPAK.
    05  TERMNAME              PIC X(88).
    05  BYPASSFEATURE         PIC S9(8) COMP.

01  SENDPAK.
    05  DONTENABLEINPUT       PIC S9(8) COMP.
    05  WINDOWENH             PIC X(8).
    05  BYPASSFEATURE         PIC S9(8) COMP.

01  FORMPAK.
    05  FORMNAME              PIC X(32).
    05  FORMPOSITION          PIC S9(8) COMP.
    05  LISTTYPE              PIC S9(8) COMP.
    05  LISTCOUNT             PIC S9(8) COMP.
    05  CHNGENTRY   OCCURS  3 TIMES.
        10  FIELD_ID          PIC X(32).
        10  CHANGE-TYPE       PIC S9(8) COMP.
        10  CHANGE-SPEC       PIC X(8).

01  MSGFORWINDOW.
    05  MSGLEN                PIC S9(8) COMP.
    05  MSGBUF.
        10  MSGAREA           PIC X(79).
        10  FILLER            PIC X(177).

01  DATADESCRPT.
    05  DESCRPTTYPE           PIC S9(8) COMP.
    05  BUFLEN                PIC S9(8) COMP.
    05  RTNBUFLEN             PIC S9(8) COMP.

01  DATABUF.
    05  DATAAREA              PIC X(200).

01  READPAK.
    05  READTIME              PIC S9(8) COMP.
    05  ENABLEREFORMAT        PIC S9(8) COMP.
    05  DOREREAD              PIC S9(8) COMP.
01  PROMPTPAK.
    05  REPAINTDATA           PIC S9(8) COMP.
    05  WINDOWENH             PIC X(8).
```

```
     05  RESETHILITED          PIC S9(8) COMP.

 01  DONE-WITH-TRANSACTIONS PIC X.

 01  ERROR-LOCATION         PIC X(70).

 01  DATA-ENTRY-ERRS        PIC X.

 01  NBR-TXN-COLLECTED      PIC 9(4).

 01  STOP-NOW               PIC X.

 01  UNUSED-PARM            PIC S9(8) COMP VALUE ZERO.

 PROCEDURE DIVISION.

 A-000-START-PROGRAM.

     MOVE "N" TO STOP-NOW
                 DONE-WITH-TRANSACTIONS.

     MOVE ZERO TO NBR-TXN-COLLECTED.

     PERFORM  A-100-SETUP-FOR-WORK.

     PERFORM  A-500-COLLECT-TRANSACTIONS
        UNTIL  STOP-NOW               = "Y"
           OR  DONE-WITH-TRANSACTIONS = "Y".

     PERFORM  A-900-CLEANUP-AFTER-WORK.

     DISPLAY " ".
     DISPLAY "Deduction transactions collected this session = "
             NBR-TXN-COLLECTED.

     IF STOP-NOW = "Y"
        PERFORM  Z-900-DISPLAY-SYSTEM-ERROR.

     STOP RUN.

 A-100-SETUP-FOR-WORK.

*****  Init Unused Parm which is used whenever intrinsic input
*****  parameter is not active.

     MOVE ZERO TO UNUSED-PARM.

***** Setup (and then forget) GlobalPak.

*****  Set Expected HP32424A Version.

     MOVE "A.00.00" TO EXPECTEDVUF OF GLOBALPAK.

*****  Set Language for COBOL.

     MOVE ZERO TO CALLPROTOCOL OF GLOBALPAK.

*****  Set Comarealen for 300 bytes.

     MOVE 300 TO COMAREALEN OF GLOBALPAK.
***** Comarea is already set to low values by value clause.

*****  Open the Transaction File.

     OPEN OUTPUT TXN-ENTRY.

*****  Open the Forms File.
```

```
     MOVE "PAYROLL.WORK.ADMIN" TO FFNAME OF FORMSFILE.

     CALL "HPDOPENFORMS" USING GLOBALPAK
                  RETURNPAK
                  FORMSFILE.

     IF RETURNSTATUS OF RETURNPAK NOT = 0
        MOVE "Y" TO STOP-NOW
        MOVE "**** Routine: Setup For Work - Forms File Open"
           TO ERROR-LOCATION
        PERFORM Z-100-GET-ERROR-MESSAGE.

***** Setup the terminal.

     IF STOP-NOW NOT = "Y"

        MOVE "HPTERM" TO TERMNAME OF TERMPAK

        MOVE ZERO TO BYPASSFEATURE OF TERMPAK

        CALL "HPDENABLETERM" USING GLOBALPAK
                      RETURNPAK
                      TERMPAK
                      UNUSED-PARM

        IF RETURNSTATUS OF RETURNPAK NOT = 0
           MOVE "Y" TO STOP-NOW
           MOVE "**** Routine: Setup For Work - Terminal Setup"
              TO ERROR-LOCATION
           PERFORM Z-100-GET-ERROR-MESSAGE.

 A-500-COLLECT-TRANSACTIONS.

*****
***** Setup for and get transaction data entry form.
*****

*****  No special Send instructions.

     MOVE ZERO TO DONTENABLEINPUT OF SENDPAK.
     MOVE SPACES TO WINDOWENH OF SENDPAK.
     MOVE ZERO TO BYPASSFEATURE OF SENDPAK.

*****  Setup to get and modify data entry form, toggling three
*****  fields to "input allowed".

     MOVE "DEDUCTION" TO FORMNAME OF FORMPAK.

*****  Position form to start at top left of display (home).

     MOVE ZERO TO FORMPOSITION OF FORMPAK.

*****  Indicate that the fields in the form which will be
*****  modified are identified by name.
     MOVE 2 TO LISTTYPE OF FORMPAK.

*****  Indicate the number of fields to modify.

     MOVE 3 TO LISTCOUNT OF FORMPAK.

*****  List fields to be modified, indicate modification type,
*****  and new value.

     MOVE "BADGE_NUMBER" TO FIELD_ID OF CHNGENTRY(1).
     MOVE 5 TO CHANGE-TYPE           OF CHNGENTRY(1).
     MOVE "O" TO CHANGE-SPEC         OF CHNGENTRY(1).

     MOVE "LAST_NAME" TO FIELD_ID    OF CHNGENTRY(2).
```

```
     MOVE 5 TO CHANGE-TYPE           OF CHNGENTRY(2).
     MOVE "O" TO CHANGE-SPEC         OF CHNGENTRY(2).

     MOVE "SUR_NAME" TO FIELD_ID     OF CHNGENTRY(3).
     MOVE 5 TO CHANGE-TYPE           OF CHNGENTRY(3).
     MOVE "O" TO CHANGE-SPEC         OF CHNGENTRY(3).

*****  Setup window message.

     MOVE 79 TO MSGLEN OF MSGFORWINDOW.

     MOVE "Fill in Deduction Transaction according to worksheet."
        TO MSGAREA OF MSGFORWINDOW.

*****  Don't copy application data out to display.

     MOVE -1 TO DESCRPTTYPE OF DATADESCRPT.

*****  Show Form.

     CALL "HPDSEND" USING GLOBALPAK
               RETURNPAK
               SENDPAK
               FORMPAK
               UNUSED-PARM
               MSGFORWINDOW
               DATADESCRPT
               UNUSED-PARM
               UNUSED-PARM
               UNUSED-PARM.

     IF RETURNSTATUS OF RETURNPAK NOT = 0
        MOVE "Y" TO STOP-NOW
        MOVE "**** Routine: Collect Transactions - Form display"
           TO ERROR-LOCATION
        PERFORM Z-100-GET-ERROR-MESSAGE.

*****  Setup and loop on transaction until it can be filed.

     MOVE "Y" TO DATA-ENTRY-ERRS.

     PERFORM B-100-READ-EDIT-AND-FILE
        UNTIL DATA-ENTRY-ERRS = "N"
           OR STOP-NOW = "Y"
           OR DONE-WITH-TRANSACTIONS = "Y".

 B-100-READ-EDIT-AND-FILE.

*****
***** Read form.
*****
*****  Enable data finishing.

     MOVE 1 TO ENABLEREFORMAT OF READPAK.

*****  No other special Read instructions.

     MOVE ZERO TO READTIME OF READPAK.
     MOVE ZERO TO DOREREAD OF READPAK.

*****  Indicate that all data in form, up to 200 bytes, is to be
*****  copied into application work space.

     MOVE 10 TO DESCRPTTYPE OF DATADESCRPT.
     MOVE 200 TO BUFLEN OF DATADESCRPT.

*****  Read form.
```

```
     CALL "HPDREAD" USING GLOBALPAK
              RETURNPAK
              READPAK
              UNUSED-PARM
              UNUSED-PARM
              DATADESCRPT
              DATABUF
              UNUSED-PARM.

     IF RETURNSTATUS OF RETURNPAK < 0
        MOVE "Y" TO STOP-NOW
        MOVE "**** Routine: Read Edit and File - Terminal Read"
           TO ERROR-LOCATION
        PERFORM Z-100-GET-ERROR-MESSAGE.

*****  Determine if operator wants to stop transaction collection.

     IF STOP-NOW NOT = "Y"
        AND RETURNSTATUS OF RETURNPAK = 0

        IF LASTITEMTYPE OF RETURNPAK = 0
           AND LASTITEMNUM OF RETURNPAK = 8
           MOVE "Y" TO DONE-WITH-TRANSACTIONS.

*****  Determine if edit errors detected.

     IF STOP-NOW NOT = "Y"
        AND DONE-WITH-TRANSACTIONS NOT = "Y"

        IF RETURNSTATUS OF RETURNPAK = 0
           MOVE "N" TO DATA-ENTRY-ERRS
        ELSE
           MOVE "Y" TO DATA-ENTRY-ERRS.

*****  Do we have a transaction that can be filed?

     IF STOP-NOW NOT = "Y"
        AND DONE-WITH-TRANSACTIONS NOT = "Y"

        IF DATA-ENTRY-ERRS NOT = "Y"
           AND LASTITEMTYPE OF RETURNPAK = 0
           AND LASTITEMNUM OF RETURNPAK = 0

*****        Write Databuf to Transaction File.

           WRITE TXN-REC FROM DATABUF

           ADD 1 TO NBR-TXN-COLLECTED.

*****  Do we need to prompt the operator to correct errors?

     IF STOP-NOW NOT = "Y"
        AND DONE-WITH-TRANSACTIONS NOT = "Y"

        IF DATA-ENTRY-ERRS = "Y"

           IF LASTITEMTYPE OF RETURNPAK = 0
              AND LASTITEMNUM OF RETURNPAK = 0

              PERFORM B-200-PROMPT-OPERATOR

           ELSE

*****           Operator pressed some key other than ENTER or
*****           EXIT so, clear data error flag to break loop
*****           (display refresh results).

              MOVE "N" TO DATA-ENTRY-ERRS.
```

```
 B-200-PROMPT-OPERATOR.

*****
***** Get message text associated with first field flagged
***** with a data error.
*****
     MOVE RETURNMSGLEN OF RETURNPAK
        TO MSGLEN OF MSGFORWINDOW.

     MOVE RETURNMSG OF RETURNPAK
        TO MSGBUF OF MSGFORWINDOW.

*****  No special Prompt instructions.

     MOVE ZERO TO REPAINTDATA OF PROMPTPAK.
     MOVE SPACES TO WINDOWENH OF PROMPTPAK.
     MOVE ZERO TO RESETHILITED OF PROMPTPAK.

*****  Display form with highlighted fields and error message
*****  in window.

     CALL "HPDPROMPT" USING GLOBALPAK
                 RETURNPAK
                 PROMPTPAK
                 UNUSED-PARM
                 MSGFORWINDOW
                 UNUSED-PARM
                 UNUSED-PARM
                 UNUSED-PARM.

     IF RETURNSTATUS OF RETURNPAK NOT = 0
        MOVE "Y" TO STOP-NOW
        MOVE "**** Routine: Prompt Operator - Display Updates"
           TO ERROR-LOCATION
        PERFORM Z-100-GET-ERROR-MESSAGE.

 A-900-CLEANUP-AFTER-WORK.

*****
***** Note that this paragraph unconditionally attempts to
***** close the Forms File and Terminal.
*****
     CLOSE TXN-ENTRY.

     CALL "HPDCLOSEFORMS" USING GLOBALPAK
                    RETURNPAK
                    FORMSFILE.

     CALL "HPDDISABLETERM" USING GLOBALPAK
                     RETURNPAK
                     TERMPAK
                     UNUSED-PARM.
 Z-100-GET-ERROR-MESSAGE.

     MOVE SPACES TO MSGAREA OF MSGBUF.

     MOVE RETURNMSG OF RETURNPAK
        TO MSGBUF.

 Z-900-DISPLAY-SYSTEM-ERROR.

     DISPLAY "**** Transaction entry facility detected system error at:".
     DISPLAY ERROR-LOCATION.
     DISPLAY "**** The error message returned is:".
     DISPLAY "**** "
              MSGAREA OF MSGBUF.
```

FORTRAN Sample Program

```
 $CONTROL list on, tables on
!
!  This application collects employee payroll deduction
!  transactions and places the edited transactions into
!  a file.
!
!  For this application:        Enter key = edit and file
!                               transaction;
!
!                                      f8 = exit application;
!
!                         all other f keys = redo transaction.
!
!  Each transaction entered by the operator is subjected to the
!  data edits embedded within the input form.
!
!  The application continues to collect transactions until either
!  the operator signals to exit or a system error is detected.
!
$TITLE '               Main Program'
!*************************************************************!
!                                                             !
!                      Main Program                           !
!                                                             !
!*************************************************************!
!
      PROGRAM FTN77EXMP
!
      IMPLICIT NONE
!
      COMMON /COM01/  GLOBALPAK
      COMMON /COM02/  RETURNPAK
      COMMON /COM03/  FORMSFILE
      COMMON /COM04/  TERMPAK
      COMMON /COM07/  MSGFORWINDOW
      COMMON /COM10/  UNUSED_PARM
      COMMON /COM101/ ERROR_LOCATION
      COMMON /COM102/ STOP_NOW
      COMMON /COM103/ DONE_WITH_TXNS
      COMMON /COM104/ NBR_TXN_COLLECTED
!
      INTEGER*4     GLOBALPAK(79)
      INTEGER*4     RETURNPAK(79)
      INTEGER*4     FORMSFILE(22)
      INTEGER*4     TERMPAK (23)
      INTEGER*4     MSGFORWINDOW(21)
      INTEGER*4     UNUSED_PARM
      CHARACTER*70  ERROR_LOCATION
      INTEGER*2     STOP_NOW
      INTEGER*2     DONE_WITH_TXNS
      INTEGER*2     NBR_TXN_COLLECTED

!
      STOP_NOW = 0
      DONE_WITH_TXNS = 0
!
      NBR_TXN_COLLECTED = 0
!
      CALL SETUP_FOR_WORK
!
      DO WHILE (STOP_NOW.EQ.0
     +    .AND.DONE_WITH_TXNS.EQ.0)
          CALL COLLECT_TXNS
      END DO
!
```

```
      CALL CLEANUP_AFTER_WORK
!
      PRINT *,
     + "Deduction transactions collected this session = ",
     + NBR_TXN_COLLECTED
!
      IF (STOP_NOW.EQ.1) THEN
         CALL DISPLAY_SYSTEM_ERROR
      END IF
!
      STOP
      END
$TITLE '              Setup For Work'
!*************************************************************!
!                                                             !
!                     Setup For Work                          !
!                                                             !
!*************************************************************!
!
      SUBROUTINE SETUP_FOR_WORK
!
      IMPLICIT NONE
!
      COMMON /COM01/  GLOBALPAK
      COMMON /COM02/  RETURNPAK
      COMMON /COM03/  FORMSFILE
      COMMON /COM04/  TERMPAK
      COMMON /COM07/  MSGFORWINDOW
      COMMON /COM10/  UNUSED_PARM
      COMMON /COM102/ STOP_NOW
      COMMON /COM101/ ERROR_LOCATION
!
      SYSTEM INTRINSIC HPDOPENFORMS,
     +                 HPDENABLETERM

!
      INTEGER*4     GLOBALPAK(79)
      CHARACTER*8   EXPECTEDVUF
      INTEGER*4     CALLPROTOCOL
      INTEGER*4     COMAREALEN
      INTEGER*4     COMAREA(75)
      EQUIVALENCE (GLOBALPAK(1), EXPECTEDVUF),
     +            (GLOBALPAK(3), CALLPROTOCOL),
     +            (GLOBALPAK(4), COMAREALEN),
     +            (GLOBALPAK(5), COMAREA)
!
      INTEGER*4     RETURNPAK(79)
      INTEGER*4     RETURNSTATUS
      EQUIVALENCE (RETURNPAK(1), RETURNSTATUS)
!
      INTEGER*4     FORMSFILE(22)
      CHARACTER*88  FFNAME
      EQUIVALENCE (FORMSFILE(1), FFNAME)
!
      INTEGER*4     TERMPAK (23)
      CHARACTER*88  TERMNAME
      INTEGER*4     TERMBYPASSFEAT
      EQUIVALENCE (TERMPAK(1), TERMNAME),
     +            (TERMPAK(23), TERMBYPASSFEAT)
!
      INTEGER*4     MSGFORWINDOW(21)
      CHARACTER*79  MSGAREA
      EQUIVALENCE (MSGFORWINDOW(2), MSGAREA)
!
      INTEGER*4     UNUSED_PARM
!
      INTEGER*2     STOP_NOW
      CHARACTER*70  ERROR_LOCATION
```

```
!
      INTEGER*2     ARRAY_INDEX
!
! Init Unused Parm which is used whenever intrinsic input
! parameter is not active.
!
      UNUSED_PARM = 0
!
! Init Comarea to all zeros.
!
      ARRAY_INDEX = 1
      DO WHILE (ARRAY_INDEX.LE.75)
         COMAREA(ARRAY_INDEX) = 0
         ARRAY_INDEX = ARRAY_INDEX + 1
      END DO
!
! Set Expected HP32424A Version.
!
      EXPECTEDVUF = "A.00.00 "

!
! Set Language for FORTRAN-77.
!
      CALLPROTOCOL = 210
!
! Set Comarealen for 300 bytes.
!
      COMAREALEN = 300
!
! Open the Transaction File:
!
      OPEN (UNIT   = 10,
     +      FILE   = 'PAYTXN',
     +      ACCESS = 'DIRECT',
     +      RECL   = 200,
     +      FORM   = 'UNFORMATTED',
     +      STATUS = 'NEW',
     +      ERR    = 110)
!
      GOTO 120
!
 110  STOP_NOW = 1
      ERROR_LOCATION =
     + "**** Routine: Setup For Work - Open Transaction File"
      MSGAREA =
     + "**** File open failed!"

!
! Open the Forms File.
!
 120  IF (STOP_NOW.EQ.0) THEN
         FFNAME = "PAYROLL.WORK.ADMIN"
!
         CALL HPDOPENFORMS (GLOBALPAK,
     +                      RETURNPAK,
     +                      FORMSFILE)
!
         IF (RETURNSTATUS.NE.0) THEN
            STOP_NOW = 1
            ERROR_LOCATION =
     +       "**** Routine: Setup For Work - Forms File Open"
            CALL UNBLOCK_MSG
         END IF
      END IF
!
! Open the Terminal.
!
      IF (STOP_NOW.EQ.0) THEN
```

```
         TERMNAME = "HPTERM"
         TERMBYPASSFEAT = 0

!
         CALL HPDENABLETERM (GLOBALPAK,
     +                       RETURNPAK,
     +                       TERMPAK,
     +                       UNUSED_PARM)
!
         IF (RETURNSTATUS.NE.0) THEN
            STOP_NOW = 1
            ERROR_LOCATION =
     +       "**** Routine: Setup For Work - Terminal Setup"
            CALL UNBLOCK_MSG
         END IF
      END IF
!
      END
$TITLE '          Collect Transactions'
!*************************************************************!
!                                                             !
!                Collect Transactions                         !
!                                                             !
!*************************************************************!
!
      SUBROUTINE COLLECT_TXNS
!
      IMPLICIT NONE
!
      COMMON /COM01/   GLOBALPAK
      COMMON /COM02/   RETURNPAK
      COMMON /COM07/   MSGFORWINDOW
      COMMON /COM08/   DATADESCRPT
      COMMON /COM10/   UNUSED_PARM
      COMMON /COM102/  STOP_NOW
      COMMON /COM103/  DONE_WITH_TXNS
      COMMON /COM104/  NBR_TXN_COLLECTED
      COMMON /COM101/  ERROR_LOCATION
      COMMON /COM105/  DATA_ENTRY_ERRS
!
      SYSTEM INTRINSIC HPDSEND
!
      INTEGER*4     GLOBALPAK(79)
!
      INTEGER*4     RETURNPAK(79)
      INTEGER*4     RETURNSTATUS
      EQUIVALENCE (RETURNPAK(1), RETURNSTATUS)
!
      INTEGER*4     MSGFORWINDOW(21)
      INTEGER*4     MSGLEN
      CHARACTER*79  MSGAREA
      EQUIVALENCE (MSGFORWINDOW(1), MSGLEN),
     +            (MSGFORWINDOW(2), MSGAREA)

!
      INTEGER*4     DATADESCRPT(3)
!
      INTEGER*4     UNUSED_PARM
!
      INTEGER*2     STOP_NOW
      INTEGER*2     DONE_WITH_TXNS
      INTEGER*2     NBR_TXN_COLLECTED
      CHARACTER*70  ERROR_LOCATION
      INTEGER*2     DATA_ENTRY_ERRS
!
      INTEGER*4     SENDPAK(4)
!
      INTEGER*4     FORMPAK(44)
```

```
      CHARACTER*32  FORMNAME
      INTEGER*4     FORMPOSITION
      INTEGER*4     LISTTYPE
      INTEGER*4     LISTCOUNT
      EQUIVALENCE (FORMPAK(1), FORMNAME),
     +            (FORMPAK(9), FORMPOSITION),
     +            (FORMPAK(10), LISTTYPE),
     +            (FORMPAK(11), LISTCOUNT)
            CHARACTER*44  FIELD_ID(1,3)
            EQUIVALENCE (FORMPAK(12), FIELD_ID)
            INTEGER*4     CHANGE_TYPE (11,3)
            CHARACTER*4   CHANGE_SPEC (11,3)
            EQUIVALENCE (FORMPAK(12), CHANGE_TYPE),
     +                  (FORMPAK(12), CHANGE_SPEC)
!
! No special Send instructions
!
      SENDPAK(1) = 0
      SENDPAK(2) = 0
      SENDPAK(3) = 0
      SENDPAK(4) = 0
!
! Setup to get and modify data entry form, toggling three
! fields to "input allowed".
!
      FORMNAME = "DEDUCTION"
!
! Position form to start at top left of display (home).
!
      FORMPOSITION = 0
!
! Indicate that the fields in the form which will be modified
! are identified by name.
!
      LISTTYPE = 2

!
! Indicate the number of fields to modify.
!
      LISTCOUNT = 3
!
! List fields to be modified, indicate modification type, and
! new value.
!
      FIELD_ID    (1,1)  = "BADGE_NUMBER"
      CHANGE_TYPE (9,1)  = 5
      CHANGE_SPEC (10,1) = "O"
!
      FIELD_ID    (1,2)  = "LAST_NAME"
      CHANGE_TYPE (9,2)  = 5
      CHANGE_SPEC (10,2) = "O"
!
      FIELD_ID    (1,3)  = "SUR_NAME"
      CHANGE_TYPE (9,3)  = 5
      CHANGE_SPEC (10,3) = "O"
!
! Setup window message.
!
      MSGLEN = 79
!
      MSGAREA =
     +  "Fill in Deduction Transaction according to worksheet."
!
! Don't copy application data out to display.
!
      DATADESCRPT(1) = -1
!
! Show form.
```

```
!
      CALL HPDSEND (GLOBALPAK,
     +              RETURNPAK,
     +              SENDPAK,
     +              FORMPAK,
     +              UNUSED_PARM,
     +              MSGFORWINDOW,
     +              DATADESCRPT,
     +              UNUSED_PARM,
     +              UNUSED_PARM,
     +              UNUSED_PARM)
!
      IF (RETURNSTATUS.NE.0) THEN
         STOP_NOW = 1
         ERROR_LOCATION =
     +    "**** Routine: Collect Transactions - Form display"
         CALL UNBLOCK_MSG
      END IF

!
! Setup and loop on transaction until it can be filed.
!
      DATA_ENTRY_ERRS = 1
!
      DO WHILE (DATA_ENTRY_ERRS.EQ.1
     +    .AND.STOP_NOW.EQ.0
     +    .AND.DONE_WITH_TXNS.EQ.0)
!
         CALL READ_EDIT_AND_FILE
!
      END DO
!
      END
$TITLE '           Read Edit and File'
!*************************************************************!
!                                                             !
!                 Read Edit and File                          !
!                                                             !
!*************************************************************!
!
      SUBROUTINE READ_EDIT_AND_FILE
!
      IMPLICIT NONE
!
      COMMON /COM01/   GLOBALPAK
      COMMON /COM02/   RETURNPAK
      COMMON /COM07/   MSGFORWINDOW
      COMMON /COM08/   DATADESCRPT
      COMMON /COM10/   UNUSED_PARM
      COMMON /COM102/  STOP_NOW
      COMMON /COM103/  DONE_WITH_TXNS
      COMMON /COM104/  NBR_TXN_COLLECTED
      COMMON /COM101/  ERROR_LOCATION
      COMMON /COM105/  DATA_ENTRY_ERRS
!
      SYSTEM INTRINSIC HPDREAD
!
      INTEGER*4     GLOBALPAK(79)
!
      INTEGER*4     RETURNPAK(79)
      INTEGER*4     RETURNSTATUS
      INTEGER*4     LASTITEMTYPE
      INTEGER*4     LASTITEMNUM
      EQUIVALENCE (RETURNPAK(1), RETURNSTATUS),
     +            (RETURNPAK(68), LASTITEMTYPE),
     +            (RETURNPAK(69), LASTITEMNUM)
!
      INTEGER*4     MSGFORWINDOW(21)
```

```
      INTEGER*4     MSGLEN
      CHARACTER*79  MSGAREA

      EQUIVALENCE (MSGFORWINDOW(1), MSGLEN),
     +            (MSGFORWINDOW(2), MSGAREA)
!
      INTEGER*4     DATADESCRPT(3)
!
      INTEGER*4     UNUSED_PARM
!
      INTEGER*2     STOP_NOW
      INTEGER*2     DONE_WITH_TXNS
      INTEGER*2     NBR_TXN_COLLECTED
      CHARACTER*70  ERROR_LOCATION
      INTEGER*2     DATA_ENTRY_ERRS
!
      INTEGER*4     READPAK(3)
      INTEGER*4     ENABLEREFORMAT
      EQUIVALENCE (READPAK(2), ENABLEREFORMAT)
!
      INTEGER*4     DATABUF(50)
      CHARACTER*200 DATAAREA
      EQUIVALENCE (DATABUF(1), DATAAREA)
!
! Enable data finishing.
!
      ENABLEREFORMAT = 1
!
! No other special Read instructions.
!
      READPAK(1) = 0
      READPAK(3) = 0
!
! Indicate that all data in form, up to 200 bytes, is to
! be copied into application work space.
!
      DATADESCRPT(1) = 10
      DATADESCRPT(2) = 200
!
! Read form.
!
      CALL HPDREAD (GLOBALPAK,
     +              RETURNPAK,
     +              READPAK,
     +              UNUSED_PARM,
     +              UNUSED_PARM,
     +              DATADESCRPT,
     +              DATABUF,
     +              UNUSED_PARM)
!
      IF (RETURNSTATUS.LT.0) THEN
         STOP_NOW = 1
         ERROR_LOCATION =
     +    "**** Routine: Read Edit and File - Terminal Read"
         CALL UNBLOCK_MSG
      END IF

!
! Determine if operator wants to stop transaction collection.
!
      IF (STOP_NOW.EQ.0
     + .AND.RETURNSTATUS.EQ.0) THEN
         IF (LASTITEMTYPE.EQ.0
     +    .AND.LASTITEMNUM.EQ.8) THEN
           DONE_WITH_TXNS = 1
         END IF
      END IF
!
```

```
! Determine if edit errors detected.
!
      IF    (STOP_NOW.EQ.0
     +  .AND.DONE_WITH_TXNS.EQ.0) THEN
!
         IF (RETURNSTATUS.EQ.0) THEN
            DATA_ENTRY_ERRS = 0
         ELSE
            DATA_ENTRY_ERRS = 1
         END IF
      END IF
!
! Do we have a transaction that can be filed?
!
      IF    (STOP_NOW.EQ.0
     +  .AND.DONE_WITH_TXNS.EQ.0) THEN
!
         IF    (DATA_ENTRY_ERRS.EQ.0
     +     .AND.LASTITEMTYPE.EQ.0
     +     .AND.LASTITEMNUM.EQ.0) THEN
!
!   Write Databuf to Transaction File.
!
            WRITE (UNIT = 10,
     +             ERR  = 310) DATAAREA
!
            GOTO 320
!
 310        STOP_NOW = 1
            ERROR_LOCATION =
     +       "**** Routine: Read Edit and File - File Write"
            MSGAREA =
     +       "**** Write to Transaction File failed!"
!
 320        IF (STOP_NOW.EQ.0) THEN
               NBR_TXN_COLLECTED = NBR_TXN_COLLECTED + 1
            END IF
         END IF
      END IF

!
! Do we need to prompt the operator to correct errors?
!
      IF (STOP_NOW.EQ.0
     +  .AND.DONE_WITH_TXNS.EQ.0) THEN
!
         IF (DATA_ENTRY_ERRS.EQ.1) THEN
            IF (LASTITEMTYPE.EQ.0
     +       .AND.LASTITEMNUM.EQ.0) THEN
!
               CALL PROMPT_OPERATOR
!
            ELSE
!
! Operator pressed some key other than ENTER or EXIT so,
! clear data error flag to break loop (display refresh results).
!
               DATA_ENTRY_ERRS = 0
!
            END IF
         END IF
      END IF
!
      END
```

```
$TITLE '               Prompt Operator'
!*****************************************************************!
!                                                                 !
!                      Prompt Operator                            !
!                                                                 !
!*****************************************************************!
!
      SUBROUTINE PROMPT_OPERATOR
!
      IMPLICIT NONE
!
      COMMON /COM01/  GLOBALPAK
      COMMON /COM02/  RETURNPAK
      COMMON /COM07/  MSGFORWINDOW
      COMMON /COM10/  UNUSED_PARM
      COMMON /COM102/ STOP_NOW
      COMMON /COM101/ ERROR_LOCATION
!
      SYSTEM INTRINSIC HPDPROMPT
!
      INTEGER*4     GLOBALPAK(79)
!
      INTEGER*4     RETURNPAK(79)
      INTEGER*4     RETURNSTATUS
      INTEGER*4     RETURNMSGLEN
      EQUIVALENCE (RETURNPAK(1), RETURNSTATUS),
     +            (RETURNPAK(3), RETURNMSGLEN)
!
      INTEGER*4     MSGFORWINDOW(21)
!
      INTEGER*4     UNUSED_PARM
!
      INTEGER*2     STOP_NOW
      CHARACTER*70  ERROR_LOCATION
!
      INTEGER*4     PROMPTPAK(4)
!
! Get error message.
!
      CALL UNBLOCK_MSG
!
!
! No special Prompt instructions.
!
      PROMPTPAK(1) = 0
      PROMPTPAK(2) = 0
      PROMPTPAK(3) = 0
      PROMPTPAK(4) = 0
!
! Display form with highlighted fields and error message
in window.
!
      CALL HPDPROMPT (GLOBALPAK,
     +                RETURNPAK,
     +                PROMPTPAK,
     +                UNUSED_PARM,
     +                MSGFORWINDOW,
     +                UNUSED_PARM,
     +                UNUSED_PARM,
     +                UNUSED_PARM)
!
      IF (RETURNSTATUS.NE.0) THEN
         STOP_NOW = 1
         ERROR_LOCATION =
     +    "**** Routine: Prompt Operator - Display Updates"
         CALL UNBLOCK_MSG
      END IF
!
```

```
      END

$TITLE '              Cleanup After Work'
!*************************************************************!
!                                                             !
!                     Cleanup After Work                      !
!                                                             !
!*************************************************************!

!
      SUBROUTINE CLEANUP_AFTER_WORK
!
      IMPLICIT NONE
!
      COMMON /COM01/ GLOBALPAK
      COMMON /COM02/ RETURNPAK
      COMMON /COM03/ FORMSFILE
      COMMON /COM04/ TERMPAK
      COMMON /COM10/ UNUSED_PARM
!
      SYSTEM INTRINSIC HPDCLOSEFORMS,
     +                 HPDDISABLETERM
!
      INTEGER*4     GLOBALPAK(79)
      INTEGER*4     RETURNPAK(79)
      INTEGER*4     FORMSFILE(22)
      INTEGER*4     TERMPAK(23)
      INTEGER*4     UNUSED_PARM
!
! Note that this routine unconditionally attempts to close
! the Forms File and Terminal
!
      CLOSE (UNIT = 10)
!
      CALL HPDCLOSEFORMS (GLOBALPAK,
     +                    RETURNPAK,
     +                    FORMSFILE)
!
! Function keys were not save thus not restored here.
!
      CALL HPDDISABLETERM (GLOBALPAK,
     +                     RETURNPAK,
     +                     TERMPAK,
     +                     UNUSED_PARM)
!
      END
$TITLE '               Unblock Message'
!*************************************************************!
!                                                             !
!                     Unblock Message                         !
!                                                             !
!*************************************************************!
!
      SUBROUTINE UNBLOCK_MSG
!
      IMPLICIT NONE
!
      COMMON /COM02/ RETURNPAK
      COMMON /COM07/ MSGFORWINDOW

!
      INTEGER*4     RETURNPAK(79)
      INTEGER*4     RETURNMSGLEN
      CHARACTER*1   RETURNMSG(254)
      EQUIVALENCE (RETURNPAK(3), RETURNMSGLEN),
     +            (RETURNPAK(4), RETURNMSG)
!
      INTEGER*4     MSGFORWINDOW(21)
```

```
      INTEGER*4     MSGLEN
      CHARACTER*1   MSGAREA(79)
      EQUIVALENCE (MSGFORWINDOW(1), MSGLEN),
     +            (MSGFORWINDOW(2), MSGAREA)
!
!
      INTEGER*2     ARRAY_INDEX
!
      ARRAY_INDEX = 1
      DO WHILE (ARRAY_INDEX.LE.RETURNMSGLEN)
         MSGAREA(ARRAY_INDEX) = RETURNMSG(ARRAY_INDEX)
         ARRAY_INDEX = ARRAY_INDEX + 1
      END DO
!
      MSGLEN = RETURNMSGLEN
!
      END
$TITLE '             Display System Error'
!*************************************************************!
!                                                             !
!                   Display System Error                      !
!                                                             !
!*************************************************************!
!
      SUBROUTINE DISPLAY_SYSTEM_ERROR
!
      IMPLICIT NONE
!
      COMMON /COM07/  MSGFORWINDOW
      COMMON /COM101/ ERROR_LOCATION
!
      INTEGER*4     MSGFORWINDOW(21)
      CHARACTER*79  MSGAREA
      EQUIVALENCE (MSGFORWINDOW(2), MSGAREA)
!
      CHARACTER*70  ERROR_LOCATION
!
      PRINT *,
     + "**** Transaction entry facility detected system error at:"
      PRINT *, ERROR_LOCATION
      PRINT *,
     + "**** The error message returned is:"
      PRINT *, MSGAREA
!
      END
```

Pascal Sample Program

```
$code_offsets on$
$tables on$

{

   This application collects employee payroll deduction
   transactions and places the edited transactions into
   a file.

   For this application:        Enter key = edit and file
                                transaction;

                                       f8 = exit application;

                           all other f keys = redo transaction.

   Each transaction entered by the operator is subjected to the
   data edits embedded within the input form.

   The application continues to collect transactions until either
   the operator signals to exit or a system error is detected.
}
PROGRAM PASCAL_EXAMPLE (OUTPUT);

TYPE
  COMAREA_TYPE          =  ARRAY [1..75] OF INTEGER;

  GLOBALPAK_REC = RECORD
    EXPECTEDVUF               :  PACKED ARRAY [1..8] OF CHAR;
    CALLPROTOCOL              :  INTEGER;
    COMAREALEN                :  INTEGER;
    COMAREA                   :  COMAREA_TYPE;
      END;

  RETURNPAK_REC = RECORD
    RETURNSTATUS              :  INTEGER;
    SUBLAYERSTATUS            :  INTEGER;
    RETURNMSGLEN              :  INTEGER;
    RETURNMSG                 :  PACKED ARRAY [1..256] OF CHAR;
    LASTITEMTYPE              :  INTEGER;
    LASTITEMNUM               :  INTEGER;
    LASTITEMNAME              :  PACKED ARRAY [1..32] OF CHAR;
    NUMDATAERRS               :  INTEGER;
    NUMCHNGFLDS               :  INTEGER;
      END;

  FORMSFILE_REC = RECORD
    FFNAME                    :  PACKED ARRAY [1..88] OF CHAR;
      END;

  TERMPAK_REC = RECORD
    TERMNAME                  :  PACKED ARRAY [1..88] OF CHAR;
    BYPASSFEATURE             :  INTEGER;
      END;

  SENDPAK_REC = RECORD
    DONTENABLEINPUT           :  INTEGER;
    WINDOWENH                 :  PACKED ARRAY [1..8] OF CHAR;
    BYPASSFEATURE             :  INTEGER;
      END;

  CHNGENTRY_REC = RECORD
    FIELD_ID                  :  PACKED ARRAY [1..32] OF CHAR;
    CHANGE_TYPE               :  INTEGER;
    CHANGE_SPEC               :  PACKED ARRAY [1..8] OF CHAR;
      END;
```

```
  FORMPAK_REC = RECORD
    FORMNAME                  :  PACKED ARRAY [1..32] OF CHAR;
    FORMPOSITION              :  INTEGER;
    LISTTYPE                  :  INTEGER;
    LISTCOUNT                 :  INTEGER;
    CHNGENTRY                 :  ARRAY [1..3] OF CHNGENTRY_REC;
      END;

  MSG_REC = RECORD
    MSGLEN                    :  INTEGER;
    MSGAREA                   :  PACKED ARRAY [1..79] OF CHAR;
      END;

  DATADESCRPT_REC = RECORD
    DESCRPTTYPE               :  INTEGER;
    BUFLEN                    :  INTEGER;
    RTNBUFLEN                 :  INTEGER;
      END;

  DATABUF_REC = RECORD
    DATAAREA                  :  PACKED ARRAY [1..200] OF CHAR;
      END;

  READPAK_REC = RECORD
    READTIME                  :  INTEGER;
    ENABLEREFORMAT            :  INTEGER;
    DOREREAD                  :  INTEGER;
      END;

  PROMPTPAK_REC = RECORD
    REPAINTDATA               :  INTEGER;
    WINDOWENH                 :  PACKED ARRAY [1..8] OF CHAR;
    RESETHILITED              :  INTEGER;
       END;

VAR
  ARRAY_INDEX            :  INTEGER;
  DATABUF                :  DATABUF_REC;
  DATADESCRPT            :  DATADESCRPT_REC;
  DATA_ENTRY_ERRS        :  BOOLEAN;
  DONE_WITH_TXNS         :  BOOLEAN;
  ERROR_LOCATION         :  PACKED ARRAY [1..70] OF CHAR;
  FORMPAK                :  FORMPAK_REC;
  FORMSFILE              :  FORMSFILE_REC;
  GLOBALPAK              :  GLOBALPAK_REC;
  MSGFORWINDOW           :  MSG_REC;
  NBR_TXNS_COLLECTED     :  INTEGER;
  PAYTXN_FILE            :  TEXT;
  PROMPTPAK              :  PROMPTPAK_REC;
  READPAK                :  READPAK_REC;
  RETURNPAK              :  RETURNPAK_REC;
  SENDPAK                :  SENDPAK_REC;
  STOP_NOW               :  BOOLEAN;
  TERMPAK                :  TERMPAK_REC;
  UNUSED_PARM            :  INTEGER;

PROCEDURE HPDOPENFORMS               ; INTRINSIC;

PROCEDURE HPDENABLETERM              ; INTRINSIC;

PROCEDURE HPDSEND                    ; INTRINSIC;

PROCEDURE HPDREAD                    ; INTRINSIC;

PROCEDURE HPDPROMPT                  ; INTRINSIC;

PROCEDURE HPDCLOSEFORMS              ; INTRINSIC;
```

```
PROCEDURE HPDDISABLETERM               ; INTRINSIC;

$TITLE '             Display System Error'$
{
*****************************************************************
*                                                               *
*                   Display System Error                        *
*                                                               *
*****************************************************************
}
PROCEDURE DISPLAY_SYSTEM_ERROR;

BEGIN

WRITELN ('Transaction entry facility detected system error at:');

WRITELN (ERROR_LOCATION);

WRITELN ('**** The error message returned is:');

WRITELN (MSGFORWINDOW.MSGAREA:MSGFORWINDOW.MSGLEN);

END;  { PROCEDURE DISPLAY_SYSTEM_ERROR }

$TITLE '               Unblock Message'$
{
*****************************************************************
*                                                               *
*                      Unblock Message                          *
*                                                               *
*****************************************************************
}
PROCEDURE UNBLOCK_MESSAGE;

BEGIN
ARRAY_INDEX := 1;

WHILE ARRAY_INDEX <= RETURNPAK.RETURNMSGLEN DO
   BEGIN
   MSGFORWINDOW.MSGAREA [ARRAY_INDEX]
      := RETURNPAK.RETURNMSG [ARRAY_INDEX];
   ARRAY_INDEX := ARRAY_INDEX + 1;
   END;

MSGFORWINDOW.MSGLEN := RETURNPAK.RETURNMSGLEN;

END;  { PROCEDURE UNBLOCK_MESSAGE }

$TITLE '              Cleanup After Work'$
{
*****************************************************************
*                                                               *
*                     Cleanup After Work                        *
*                                                               *
*****************************************************************
}
PROCEDURE CLEANUP_AFTER_WORK;

BEGIN

{ Note that this routine unconditionally attempts to close
  the Forms File and Terminal.                                 }

CLOSE (PAYTXN_FILE,
       'SAVE');

HPDCLOSEFORMS (GLOBALPAK,
               RETURNPAK,
```

```
               FORMSFILE);

HPDDISABLETERM (GLOBALPAK,
                RETURNPAK,
                TERMPAK,
                UNUSED_PARM);

END;  { PROCEDURE CLEANUP_AFTER_WORK }

$TITLE '               Prompt Operator'$
{
*****************************************************************
*                                                               *
*                      Prompt Operator                          *
*                                                               *
*****************************************************************
}
PROCEDURE PROMPT_OPERATOR;

BEGIN

{ Get error message. }

UNBLOCK_MESSAGE;

{ No special Prompt instructions. }

PROMPTPAK.REPAINTDATA  := 0;
PROMPTPAK.WINDOWENH    := ;
PROMPTPAK.RESETHILITED := 0;

{ Display form with highlighted fields and error message
  in window.                                                  }

HPDPROMPT (GLOBALPAK,
           RETURNPAK,
           PROMPTPAK,
           UNUSED_PARM,
           MSGFORWINDOW,
           UNUSED_PARM,
           UNUSED_PARM,
           UNUSED_PARM);

IF RETURNPAK.RETURNSTATUS
0 THEN
   BEGIN
   STOP_NOW := TRUE;
   ERROR_LOCATION :=
      '**** Routine: Prompt Operator - Display Updates';
   UNBLOCK_MESSAGE;
   END;

END;  { PROCEDURE PROMPT_OPERATOR }

$TITLE '           Read Edit and File'$
{
!*************************************************************!
!                                                             !
!                 Read Edit and File                          !
!                                                             !
!*************************************************************!
}
PROCEDURE READ_EDIT_AND_FILE;

BEGIN

{ Enable data finishing. }
```

```
READPAK.ENABLEREFORMAT := 1;

{ No other special Read instructions. }

READPAK.READTIME := 0;
READPAK.DOREREAD := 0;

{ Indicate that all data in form, up to 200 bytes, is to be
  copied into application work space.                      }

DATADESCRPT.DESCRPTTYPE := 10;
DATADESCRPT.BUFLEN      := 200;

{ Read form. }

HPDREAD (GLOBALPAK,
         RETURNPAK,
         READPAK,
         UNUSED_PARM,
         UNUSED_PARM,
         DATADESCRPT,
         DATABUF,
         UNUSED_PARM);

IF RETURNPAK.RETURNSTATUS < 0 THEN
   BEGIN
   STOP_NOW := TRUE;
   ERROR_LOCATION :=
      '**** Routine: Read Edit and File - Terminal Read';
   UNBLOCK_MESSAGE;
   END;

{ Determine if operator wants to stop transaction collection. }

IF NOT STOP_NOW
   AND (RETURNPAK.RETURNSTATUS = 0) THEN

   IF (RETURNPAK.LASTITEMTYPE = 0)
      AND (RETURNPAK.LASTITEMNUM = 8) THEN
      DONE_WITH_TXNS := TRUE;

{ Determine if edit errors detected. }

IF NOT STOP_NOW
   AND NOT DONE_WITH_TXNS THEN

   IF RETURNPAK.RETURNSTATUS = 0 THEN
      DATA_ENTRY_ERRS := FALSE
   ELSE
      DATA_ENTRY_ERRS := TRUE;

{ Do we have a transaction that can be filed? }

IF NOT STOP_NOW
   AND NOT DONE_WITH_TXNS THEN

   IF NOT DATA_ENTRY_ERRS
      AND (RETURNPAK.LASTITEMTYPE = 0)
      AND (RETURNPAK.LASTITEMNUM  = 0) THEN
      BEGIN

      { Write Databuf to Transaction File. }

      WRITELN (PAYTXN_FILE,
               DATABUF.DATAAREA : DATADESCRPT.RTNBUFLEN);

      NBR_TXNS_COLLECTED := NBR_TXNS_COLLECTED + 1;
```

```
      END;

{ Do we need to prompt the operator to correct errors? }

IF NOT STOP_NOW
   AND NOT DONE_WITH_TXNS THEN

   IF DATA_ENTRY_ERRS THEN

      IF (RETURNPAK.LASTITEMTYPE = 0)
         AND (RETURNPAK.LASTITEMNUM = 0) THEN

         PROMPT_OPERATOR

      ELSE

         { Operator pressed some key other than ENTER or
           EXIT so, clear data error flag to break loop
           (display refresh results).                    }

         DATA_ENTRY_ERRS := FALSE;

END;  { PROCEDURE READ_EDIT_AND_FILE }

$TITLE '          Collect Transactions'$
{
!************************************************************!
!                                                            !
!                Collect Transactions                        !
!                                                            !
!************************************************************!
}
PROCEDURE COLLECT_TRANSACTIONS;

BEGIN

{ No special Send instructions. }

SENDPAK.DONTENABLEINPUT := 0;
SENDPAK.WINDOWENH       := '';
SENDPAK.BYPASSFEATURE   := 0;

{ Setup to get and modify data entry form, toggling three
  fields to 'input allowed'.                              }

FORMPAK.FORMNAME := 'DEDUCTION';

{ Position form to start at top left of display (home). }

FORMPAK.FORMPOSITION := 0;

{ Indicate that the fields in the form which will be modified
  are identified by name.                                   }

FORMPAK.LISTTYPE := 2;

{ Indicate the number of fields to modify. }

FORMPAK.LISTCOUNT := 3;

{ List fields to be modified, indicate modification type, and
  new value.                                                 }

FORMPAK.CHNGENTRY [1].FIELD_ID := 'BADGE_NUMBER';
FORMPAK.CHNGENTRY [1].CHANGE_TYPE := 5;
FORMPAK.CHNGENTRY [1].CHANGE_SPEC := 'O';

FORMPAK.CHNGENTRY [2].FIELD_ID := 'LAST_NAME';
```

```
FORMPAK.CHNGENTRY [2].CHANGE_TYPE := 5;
FORMPAK.CHNGENTRY [2].CHANGE_SPEC := 'O';

FORMPAK.CHNGENTRY [3].FIELD_ID := 'SUR_NAME';
FORMPAK.CHNGENTRY [3].CHANGE_TYPE := 5;
FORMPAK.CHNGENTRY [3].CHANGE_SPEC := 'O';

{ Setup window message. }

MSGFORWINDOW.MSGLEN := 79;

MSGFORWINDOW.MSGAREA :=
   'Fill in Deduction Transaction according to worksheet.';

{ Don't copy application data out to display. }

DATADESCRPT.DESCRPTTYPE := -1;

{ Show Form. }

HPDSEND (GLOBALPAK,
         RETURNPAK,
         SENDPAK,
         FORMPAK,
         UNUSED_PARM,
         MSGFORWINDOW,
         DATADESCRPT,
         UNUSED_PARM,
         UNUSED_PARM,
         UNUSED_PARM);

IF RETURNPAK.RETURNSTATUS
0 THEN
   BEGIN
   STOP_NOW := TRUE;
   ERROR_LOCATION :=
      '**** Routine: Collect Transactions - Form display';
   UNBLOCK_MESSAGE;
   END;

{ Setup and loop on transaction until it can be filed. }
DATA_ENTRY_ERRS := TRUE;

WHILE DATA_ENTRY_ERRS
  AND NOT STOP_NOW
  AND NOT DONE_WITH_TXNS DO
   READ_EDIT_AND_FILE;

END;  { PROCEDURE COLLECT_TRANSACTIONS }

$TITLE '              Setup For Work'$
{
*****************************************************************
*                                                               *
*                     Setup For Work                            *
*                                                               *
*****************************************************************
}
PROCEDURE SETUP_FOR_WORK;

BEGIN

{ Init Unused Parm which is used whenever intrinsic input
  parameter is not active.                                   }

UNUSED_PARM := 0;

{ Init Comarea to all zeros. }
```

```
ARRAY_INDEX := 1;
WHILE ARRAY_INDEX <= 75 DO
   BEGIN
   GLOBALPAK.COMAREA [ARRAY_INDEX] := 0;
   ARRAY_INDEX := ARRAY_INDEX + 1;
   END;

{ Set Expected HP32424A Version. }

GLOBALPAK.EXPECTEDVUF := 'A.00.00';

{ Set Language for Pascal. }

GLOBALPAK.CALLPROTOCOL := 500;

{ Set Comarealen for 300 bytes. }

GLOBALPAK.COMAREALEN := 300;

{ Open the Transaction File. }

APPEND (PAYTXN_FILE,
        'PAYTXN');
{ Open the Forms File. }

FORMSFILE.FFNAME := 'PAYROLL.WORK.ADMIN';

HPDOPENFORMS (GLOBALPAK,
              RETURNPAK,
              FORMSFILE);

IF RETURNPAK.RETURNSTATUS
0 THEN
   BEGIN
   STOP_NOW := TRUE;
   ERROR_LOCATION :=
      '**** Routine: Setup For Work - Forms File Open';
   UNBLOCK_MESSAGE;
   END;

IF NOT STOP_NOW THEN
   BEGIN

   TERMPAK.TERMNAME := 'HPTERM';

   TERMPAK.BYPASSFEATURE := 0;

   HPDENABLETERM (GLOBALPAK,
                  RETURNPAK,
                  TERMPAK,
                  UNUSED_PARM);

   IF RETURNPAK.RETURNSTATUS
0 THEN
      BEGIN
      STOP_NOW := TRUE;
      ERROR_LOCATION :=
         '**** Routine: Setup For Work - Terminal Setup';
      UNBLOCK_MESSAGE;
      END;

   END;

END;  { PROCEDURE SETUP_FOR_WORK }
```

```
$TITLE '               Main Program'$
{
*****************************************************************
*                                                               *
*                      Main Program                             *
*                                                               *
*****************************************************************
}
BEGIN

STOP_NOW := FALSE;
DONE_WITH_TXNS := FALSE;

NBR_TXNS_COLLECTED := 0;

SETUP_FOR_WORK;

WHILE NOT STOP_NOW
  AND NOT DONE_WITH_TXNS DO

   COLLECT_TRANSACTIONS;

CLEANUP_AFTER_WORK;
WRITELN ('Deduction transactions collected this session = ',
         NBR_TXNS_COLLECTED);

IF STOP_NOW THEN
   DISPLAY_SYSTEM_ERROR;

END.  { Main Program }"
```

Appendix D Terminals Supported by HI-LI

This appendix contains information about those terminals that are supported by the Hi-Li intrinsics, and discusses how Hi-Li uses their various features. In addition, sections about terminal buffer configuration and how to recover from unexpected program interruptions are provided. Refer to this appendix if you have any terminal dependent questions as you develop your application.

Supported Terminals

Hi-Li supports the following terminals:

HP 150	HP 2392A	HP 2622A
	HP 2393A	HP 2623A
HP 2382A	HP 2394A	HP 2624A
	HP 2397A	HP 2624B
		HP 2625A
		HP 2626A
		HP 2627A
		HP 2628A

For information about the terminal features that Hi-Li can use, see "Supported Features" in this appendix.

The HP 239X and HP 150 Terminals

You can run Hi-Li on the HP 239X and HP 150 terminals without specifying HP 262X or HP 239X on the FORMSPEC Terminal/Language Selection Menu except for special cases that are documented below.

To use the X.25 block mode via the PAD interface that is available on all HP 239X and HP 150 terminals, the transmit and receive pacing must be set correctly and the terminal must be strapped correctly. (The HP 239X and HP 150 terminals will always have the correct ROMS installed). You must also specify the correct terminal type when you log on to the terminal. For complete information about using the X.25 block mode feature, see the
DSN/X.25 HP 3000 Reference Manual (Part No. 32191-90001)

The HP 150 Terminal. To use the security display enhancement feature for this terminal, you must specify the HP 262X or HP 239X family of

terminals on FORMSPEC's Terminal/Language Selection Menu.

The HP 2392A Terminal. To use the security display enhancement feature on this terminal, you must specify the HP 262X and HP 239X family of terminals on FORMSPEC's Terminal/Language Selection Menu.

The HP 2393A Terminal. To use the security display enhancement feature on this terminal, you must specify the HP 262X and HP 239X family of terminals on FORMSPEC's Terminal/Language Selection Menu.

The HP 2394A Terminal. To use the local edits and the security display enhancement features of this terminal, you must specify the HP 262X and HP 239X family of terminals on FORMSPEC's Terminal/Language Selection Menu.

The HP 2397A Terminal. To use the color display enhancement feature of this terminal, you must specify both the HP 262X and HP 239X and the HP 2627 and HP 2397A families of terminals on FORMSPEC's Terminal/Language Selection Menu.

The HP 262X Terminals

You can run Hi-Li on the HP 262X family of terminals without specifying HP 262X or HP 239X on the FORMSPEC Terminal/Language Selection Menu except for special cases that are documented below.

The X.25 block mode feature is available via a PAD interface with all HP 262X terminals except the HP 2624A. To use this feature, you must be sure that the transmit and receive pacing is set correctly, the appropriate ROMS are used, and the terminal is strapped correctly. The HP 2625A and HP 2628A terminals will always have the correct ROMS. In addition, you must use the correct terminal type when you log on to the terminal. For complete information about using the X.25 block mode feature, see the *DSN/X.25 HP 3000 Reference Manual* (Part No. 32191-90001).

The HP 2624 Terminal. To use the local edits and security display enhancement features on this terminal, you must specify the HP 262X and HP 239X families of terminals on the FORMSPEC Terminal/Language Selection Menu.

When using a form, the terminal can issue the following message at run time:

```
LINE IS FULL - RETURN TO CLEAR.
```

This message means that there are too many characters on the line. You must redesign and recompile the form.

The HP 2624B Terminal. To use the local edits and security display enhancement features on this terminal, you must specify the HP 262X and HP 239X families of terminals on the FORMSPEC Terminal/Language Selection Menu.

When using a form, the terminal can issue the following message at run time:

LINE IS FULL - RETURN TO CLEAR.

This message means that there are too many characters on the line. You must redesign and recompile the form.

The HP 2625A and HP 2628A Terminals. To use the security display enhancement feature of these terminals, you must specify the HP 262X and HP 239X families of terminals on the FORMSPEC Terminal/Language Selection Menu.

The HP 2626A Terminal. To use the security display enhancement feature of this terminal, you must specify the HP 262X and HP 239X families of terminals on the FORMSPEC Terminal/Language Selection Menu.

The HP 2627A Terminal. To use the color display enhancement feature of this terminal, you must specify both the HP 262X and HP 239X and the HP 2627 and HP 2397A families of terminals on FORMSPEC's Terminal/Language Selection Menu.

Supported Features

Table D-1 shows those features of the supported terminals that can be used by Hi-Li. A discussion of each of the features follows the table.

Table D-1. Terminal Features Used by Hi-Li

ID	MODEL	GRAPH	COLOR	TOUCH	MDT	LOCAL EDITS	SEC ENH	X.25 PAD
7*	2621A							
8	2626A						X	
9	2624A				X	X	X	
11	2622A							X
12	2623A	X						X
13	2624B				X	X	X	X
14	2382A							
18*	2621B							
52	2627A	X	X					X
55	2622E							
56	2392A							X
57	2394A				X	X	X	X
61	2625A				X		X	X
62	2628A				X		X	X
63	2625A	X			X		X	X
64	2628A	X			X		X	X
65	2393A	X			X		X	X
66	2397A	X	X		X			X
67	2393A	X		X	X		X	X
68	2397A	X	X	X	X			X
70	HP150	X		X	X		X	X

* Not a supported terminal; they are recognized by Hi-Li, but not used.

Modified Data Tag (MDT)

This feature allows you to specify that only the fields that have been modified are transmitted to the computer. You do not need to take any action to use this feature.

Extended Local Edits

This feature allows the terminal to edit information as it is typed onto the form. You specify the editing you want to be done by entering the LOCALEDITS command in the field processing specification section of FORMSPEC.

Relabelling Function Keys

This feature allows you to specify, at form creation or modification time, those function key labels that will be used during data entry. Hi-Li can perform function key labeling on all the terminals it supports.

Security Display Enhancement (SEC ENH)

This feature allows you to inhibit the display of data in certain fields. When this feature is used, the characters entered at the terminal are displayed as blanks on the screen.

X.25 Capability

This feature allows X.25 block mode to be used via a PAD interface. When you use this capability on the HP 2622A, HP 2623A, and HP 2624B terminals, check the ROMS for auto keyboard lock.

Color Enhancement

This feature allows you to use color for field, error, and window enhancements with the HP 2397A and HP 2627A terminals.

Terminal Buffer Configuration

When using the HPDSEND, HPDREAD, or HPDPROMPT intrinsics, your system must have sufficient terminal buffers available for all concurrently executing terminal I/O operations. The number of terminal buffers on your system must be at least 150 and it is recommended that you set the number of terminal buffers to the maximum number shared by all processes that your system allows. For information about terminal buffers, see the configuration dialogue in the *MPE/V Systems Operation and Resource Management Reference Manual* (Part No. 32033-90005).

Recovering From Unexpected Program Interruption

A robust application design that uses the Hi-Li intrinsics will include a REFRESH function key for every form. The application would implement a

REFRESH function key by calling the HPDSEND intrinsic and passing the $REFRESH form name token when a REFRESH key request is detected. With a design that includes a REFRESH function key, the following recovery procedures should be used when an unexpected program interruption occurs.

If the interruption occurs because the BREAK key was pressed or because of a terminal power failure, control returns to MPE. To recover from this situation, take the following steps:

1. Do a hard reset by pressing the SHIFT, CONTROL, and RESET keys simultaneously. Then, press [Return] to display the colon prompt.

2. If echo is turned off, press the [ESC] key, then the colon (:) key to restore echo.

3. Type RESUME. The message READ PENDING is displayed.

4. Press the [REFRESH] key to return to the menu at which you were interrupted.